

图解CSS3

核心技术与案例实战

大漠 著

Illustrated CSS3
Details and Cases

- 资深Web前端专家历时两载的经验与心血之作，旨在根据最新CSS3规范撰写最权威的CSS3学习资料和备查手册
- 理论知识系统且全面，以图解的方式讲解CSS3的各项功能和特性，包含大量实战案例，直观易懂，实战性强



CSS3在前端开发中的重要性毋庸置疑，这些年来，新的CSS3规范在不断演进和完善，但是一直没有确定的规范。从技术标准的角度来讲，本书应该是目前版本最新的；从知识点的涵盖面来讲，本书也是同类书中最全面的。更难得的是，为了便于读者理解，作者用大量直观的图示替代了枯燥的文字，采用了图解的方式来讲解，相信这应该会很受读者欢迎。此外，本书还包含大量实战案例，理论与实践相结合。如果你要系统学习CSS3或者在开发中还不能熟练使用它，强烈推荐这本书给你。

——51CTO (www.51cto.com) 中国领先的IT技术网站

CSS在Web前端中一直占有非常重要的位置，因为它与产品的用户体验直接相关。CSS3规范的起草和修订持续了10余年，至今仍未结束，更具吸引力的新特性不断出现，也不断有特性被修改和完善，CSS3规范一直在变化之中，这使得原本就相对复杂的学习任务变得难上加难。本书作者大漠作为一位资深的Web前端技术专家，在CSS3领域浸淫多年，不仅见证了CSS3规范的发展和演变过程，而且经历了几乎所有CSS3的初学者可能会遇到的各种问题，包括技术上的，也包括学习方法上的。为了让后来者少走弯路，大漠根据最新的CSS3规范，结合自己的学习经历，把自己这些年来在CSS3上的实践经验以“图解+案例”的形式集中呈现在这本书中，希望能帮助所有读者在系统学习和使用CSS3的过程中达到事半功倍的效果。



上架指导：计算机/Web开发

ISBN 978-7-111-46920-9



9 787111 469209 >

定价：79.00元

图解CSS3

核心技术与案例实战

Illustrated CSS3
Details and Cases

大漠 著



机械工业出版社
China Machine Press

图书在版编目 (CIP) 数据

图解 CSS3: 核心技术与案例实战 / 大漠著. —北京: 机械工业出版社, 2014.7
(Web 开发技术丛书)

ISBN 978-7-111-46920-9

I. 图… II. 大… III. 网页制作工具 IV. TP393.092

中国版本图书馆 CIP 数据核字 (2014) 第 116144 号

图解 CSS3: 核心技术与案例实战

大 漠 著

出版发行: 机械工业出版社 (北京市西城区百万庄大街 22 号 邮政编码: 100037)

责任编辑: 杨福川

责任校对: 殷 虹

印 刷: 三河市宏图印务有限公司

版 次: 2014 年 7 月第 1 版第 1 次印刷

开 本: 186mm×240mm 1/16

印 张: 32.25 (含彩插 1.25 印张)

书 号: ISBN 978-7-111-46920-9

定 价: 79.00 元

凡购本书, 如有缺页、倒页、脱页, 由本社发行部调换

客服热线: (010) 88378991 88361066

投稿热线: (010) 88379604

购书热线: (010) 68326294 88379649 68995259

读者信箱: hzjsj@hzbook.com

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问: 北京大成律师事务所 韩光 / 邹晓东

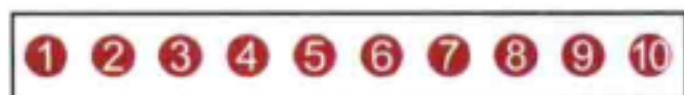


图 2-2 页面初步效果



图 2-3 通配选择器使用效果



图 2-4 元素选择器使用效果

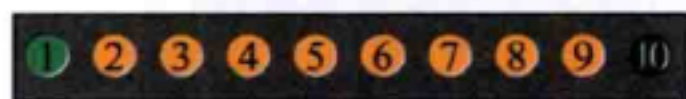


图 2-5 ID 选择器使用效果



图 2-6 类选择器使用效果



图 2-7 多类名选择器使用效果

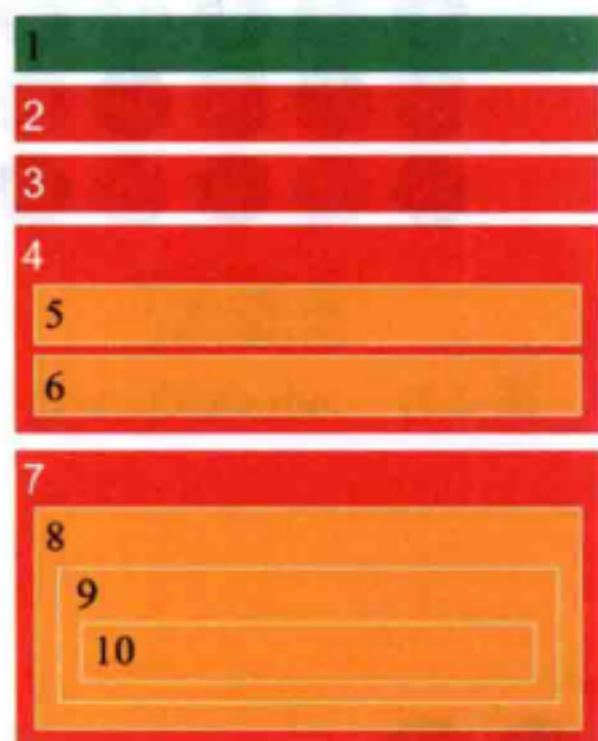


图 2-13 使用通用兄弟选择器效果



图 2-14 美化按钮效果



图 2-15 IE 8 下的按钮效果

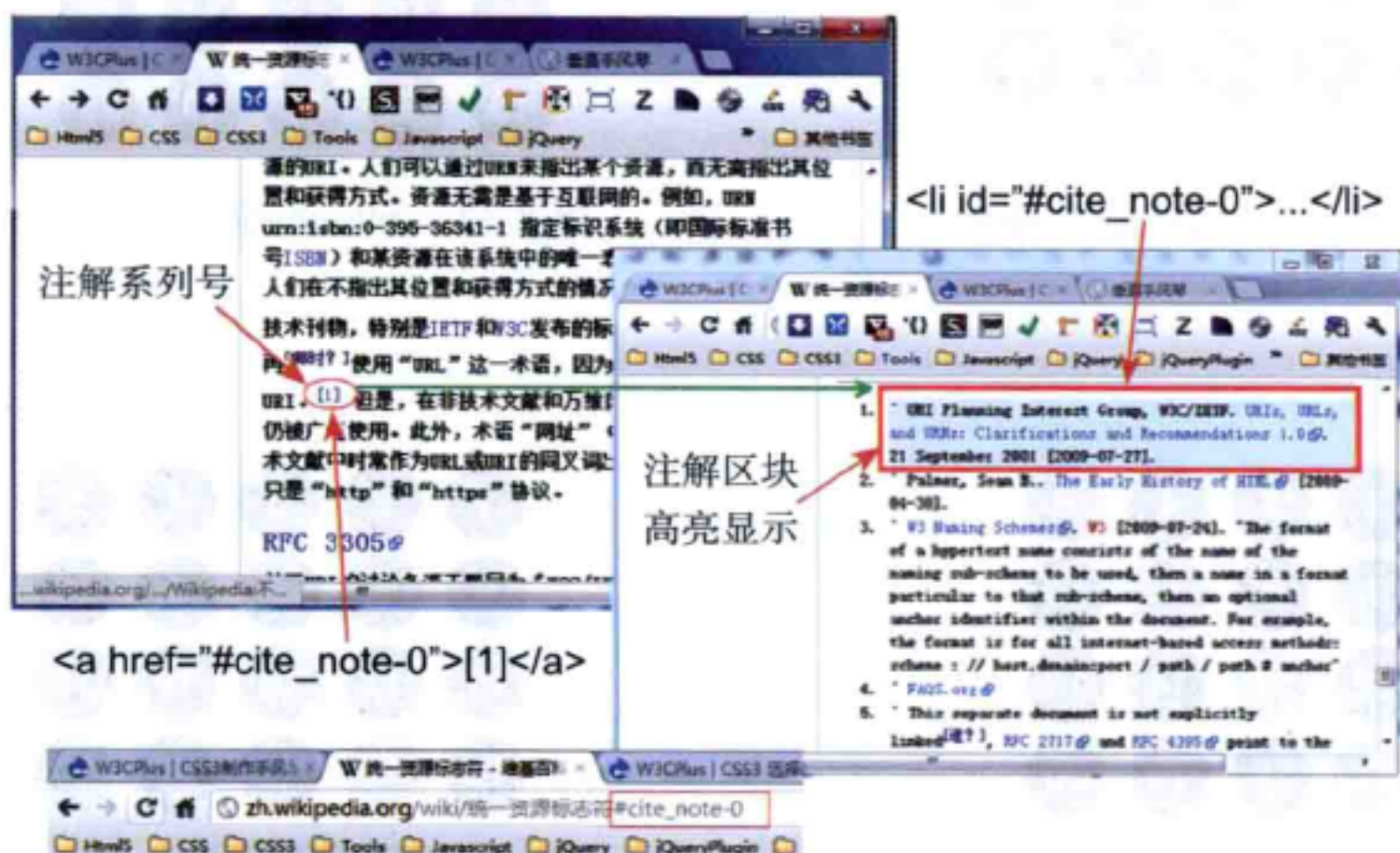


图 2-17 目标伪类选择器高亮显示区块的运用

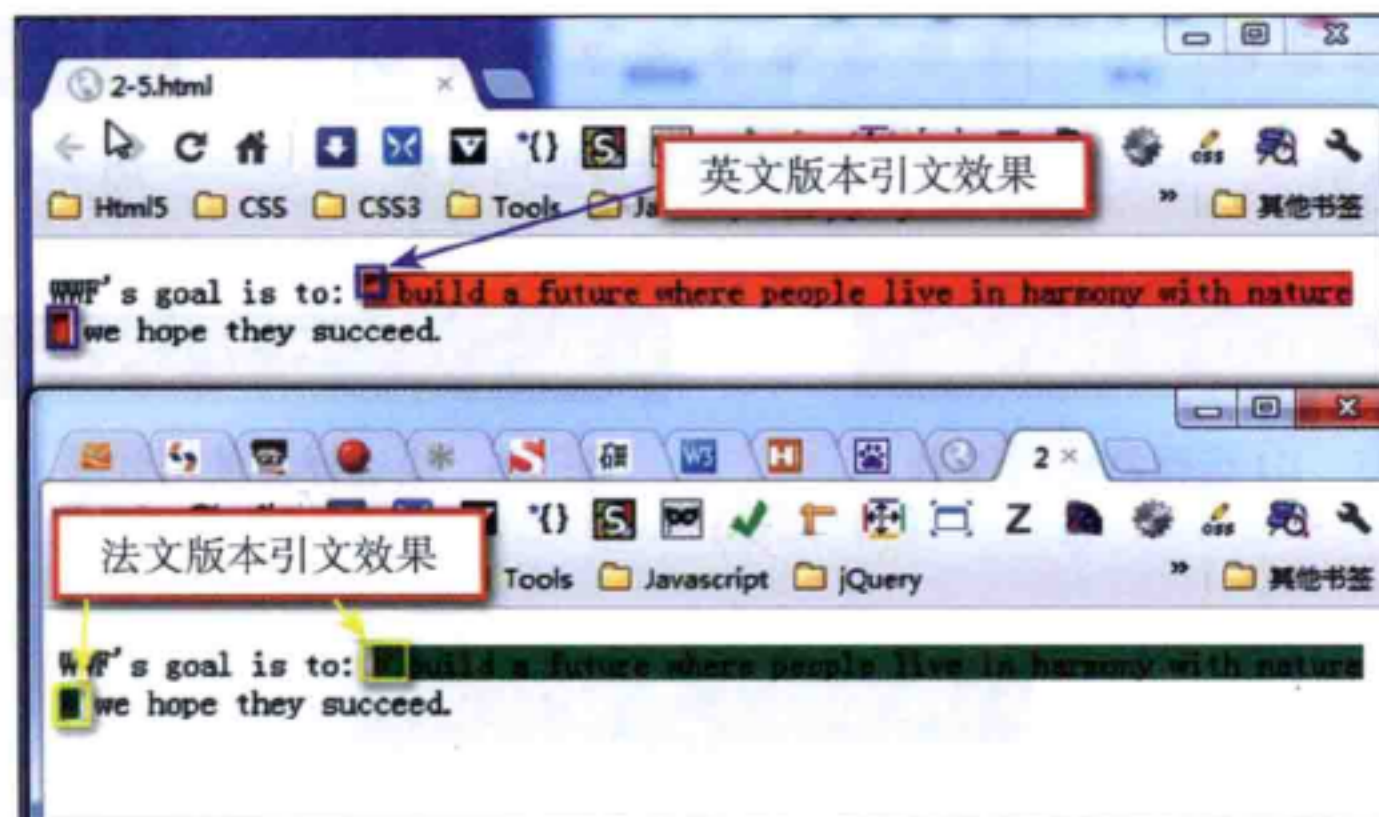


图 2-19 多语言版本引文的效果



图 2-29 :nth-child(n) 效果

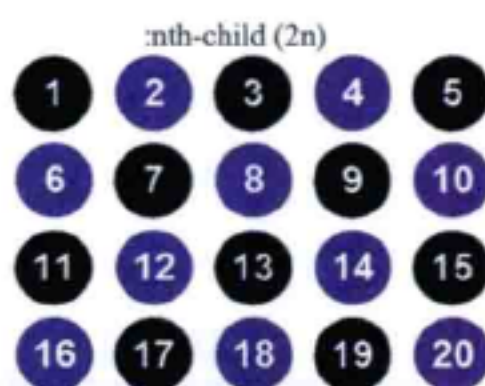


图 2-30 :nth-child(2n) 效果

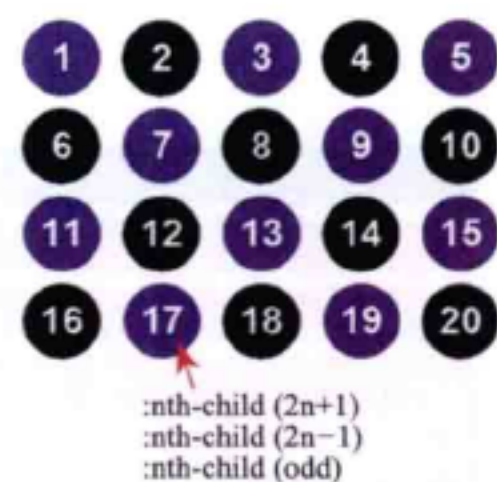


图 2-31 :nth-child(2n+1) 效果

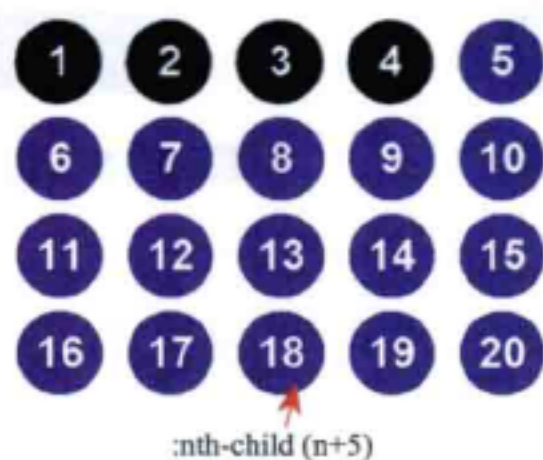


图 2-32 :nth-child(n+5) 效果

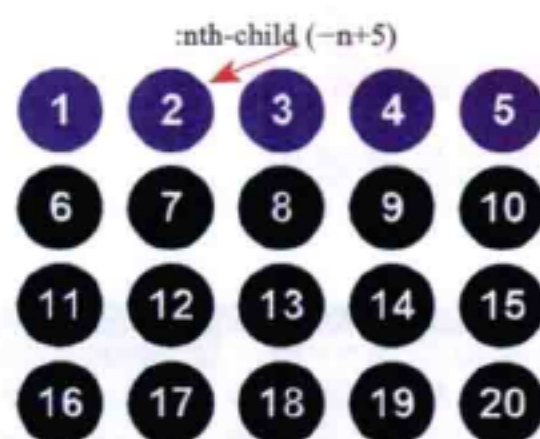


图 2-34 :nth-child(-n+5) 效果

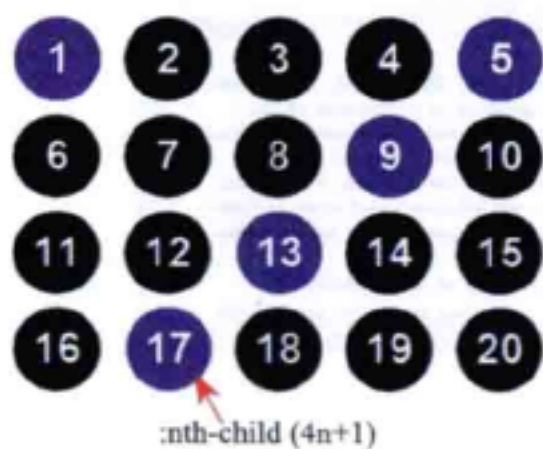


图 2-36 :nth-child(4n+1) 效果



图 2-37 :nth-last-child(4) 效果

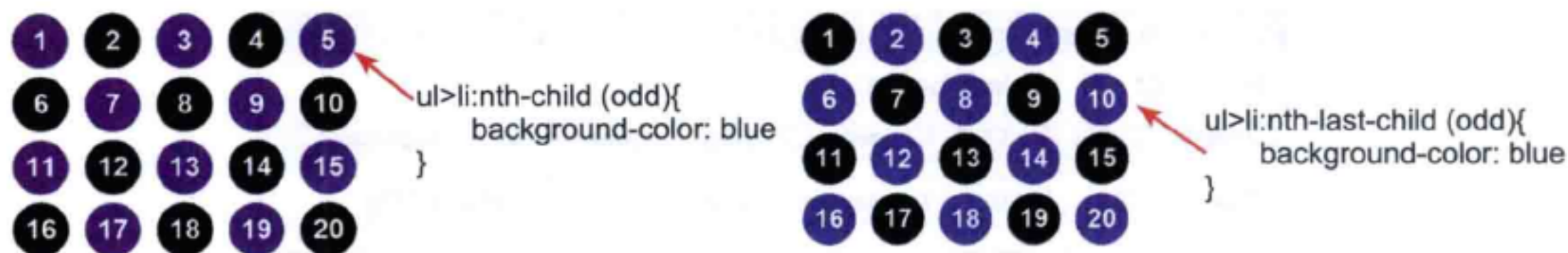


图 2-38 :nth-child(odd) 与 :nth-last-child(odd) 对比



图 2-39 :nth-child(even) 和 :nth-last-child(even) 对比



图 3-15 各浏览器下 border-image 制作按钮效果



图 3-17 各浏览器下 border-image 制作 tabs 效果

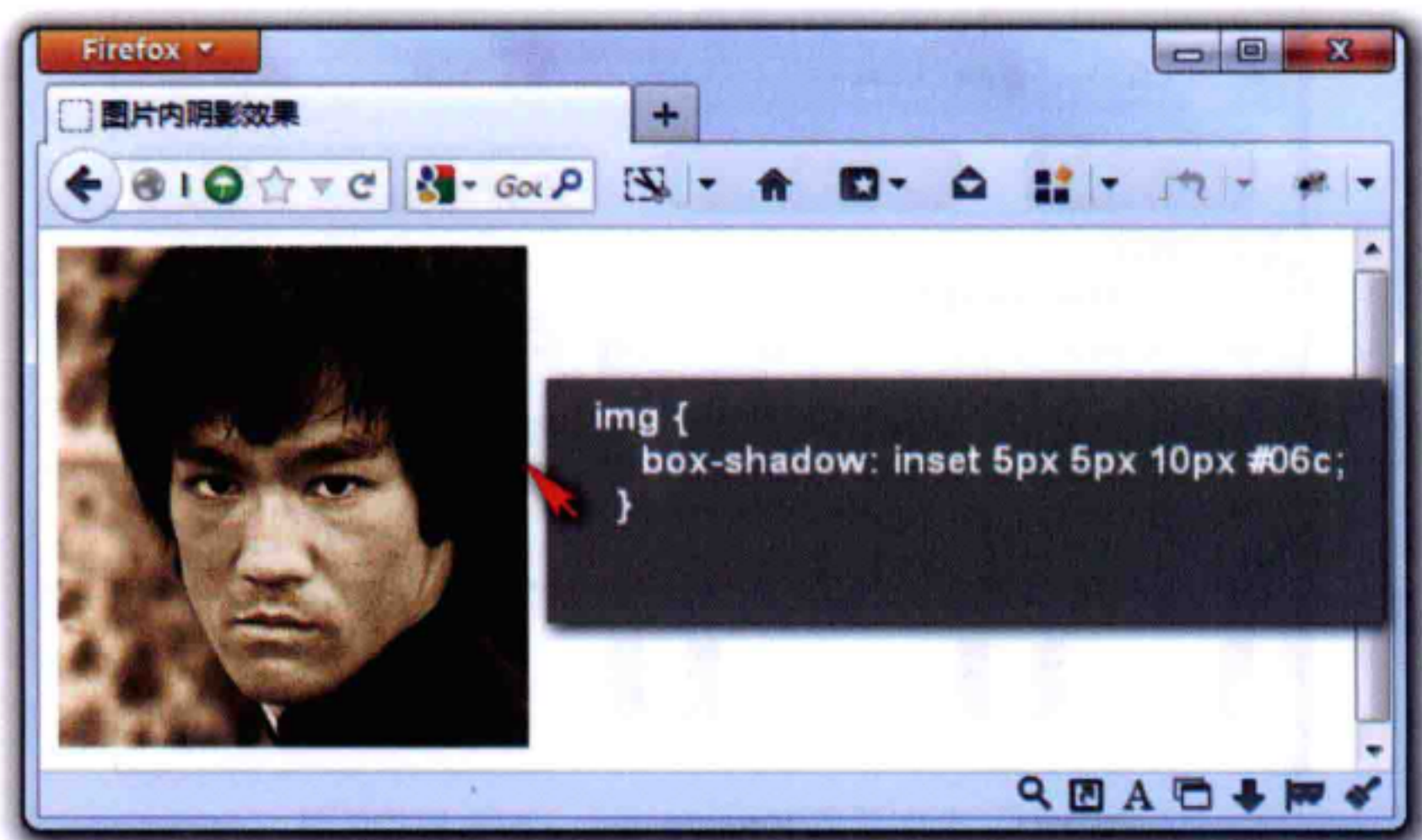


图 3-48 box-shadow 在 img 上的内阴影无效果

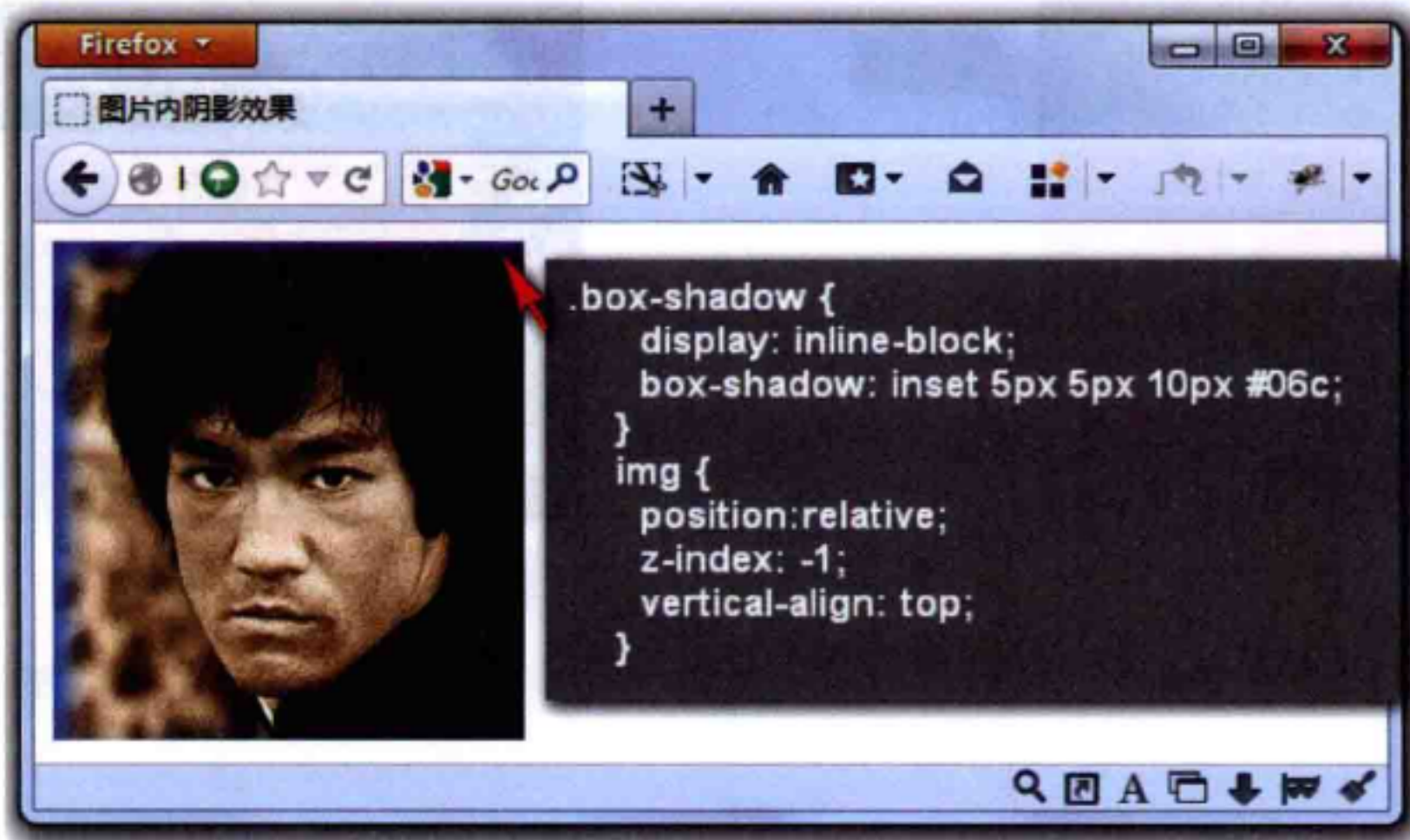


图 3-49 图片内阴影效果

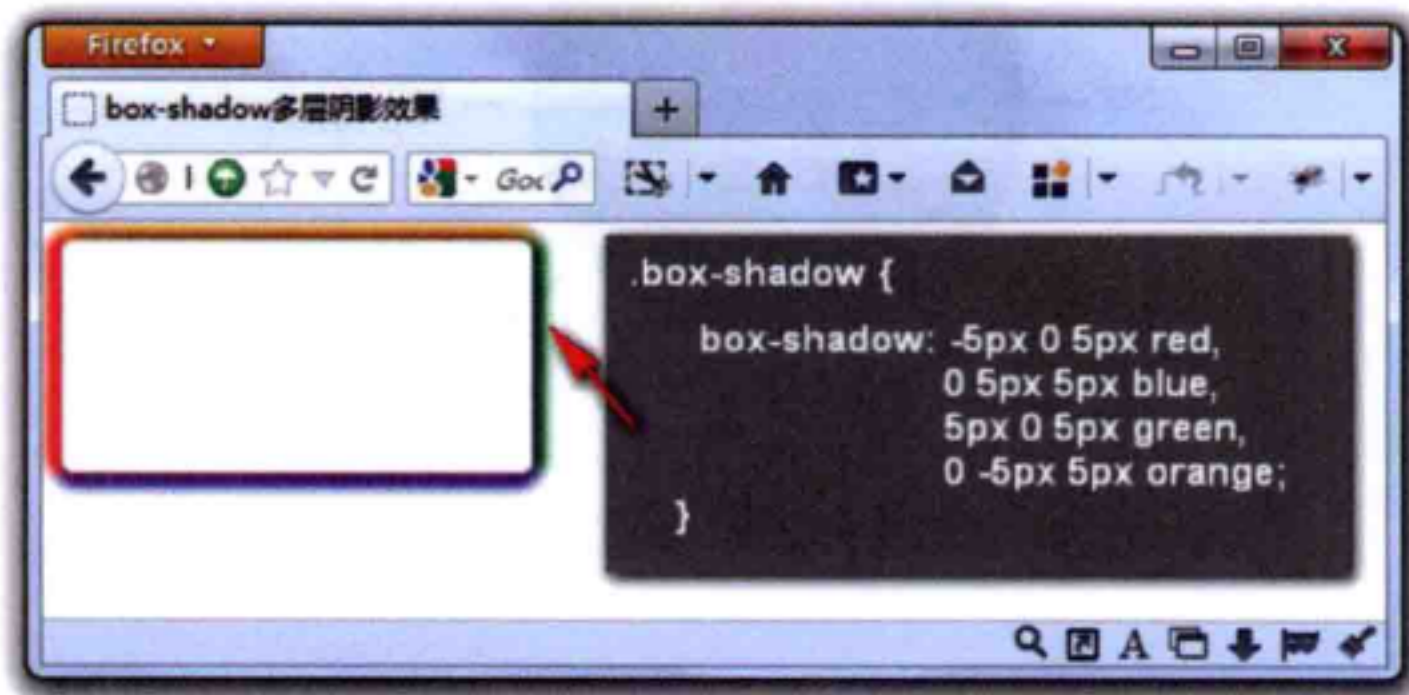


图 3-50 box-shadow 多层阴影效果



图 4-22 CSS3 多背景制作花边框



图 4-23 五张背景图切片

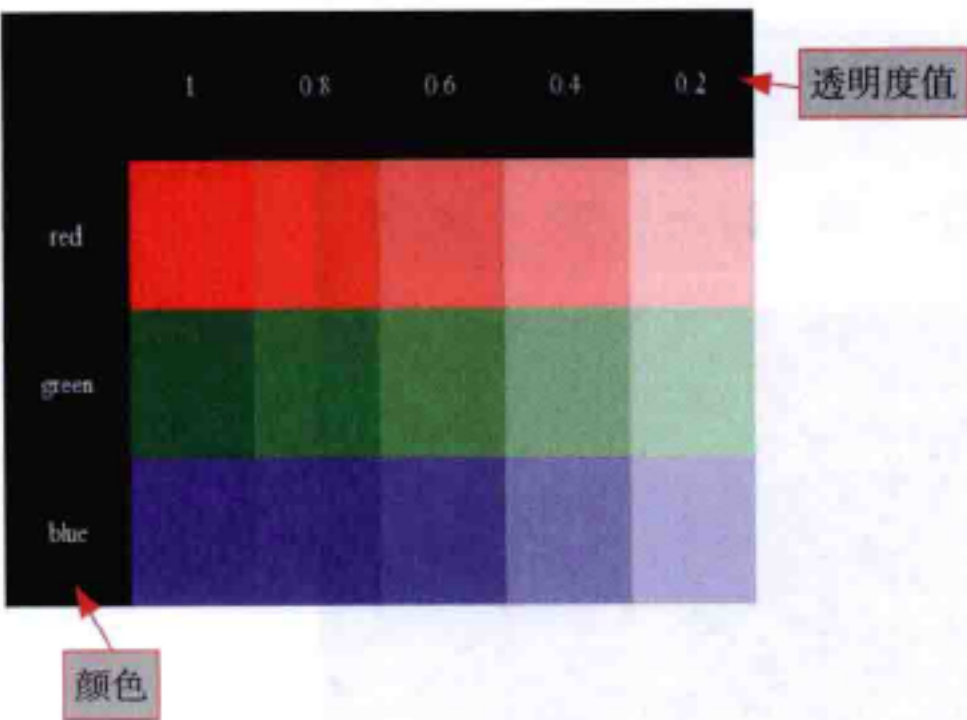


图 6-2 通过 opacity 制作渐变色块 (Chrome 25 下)

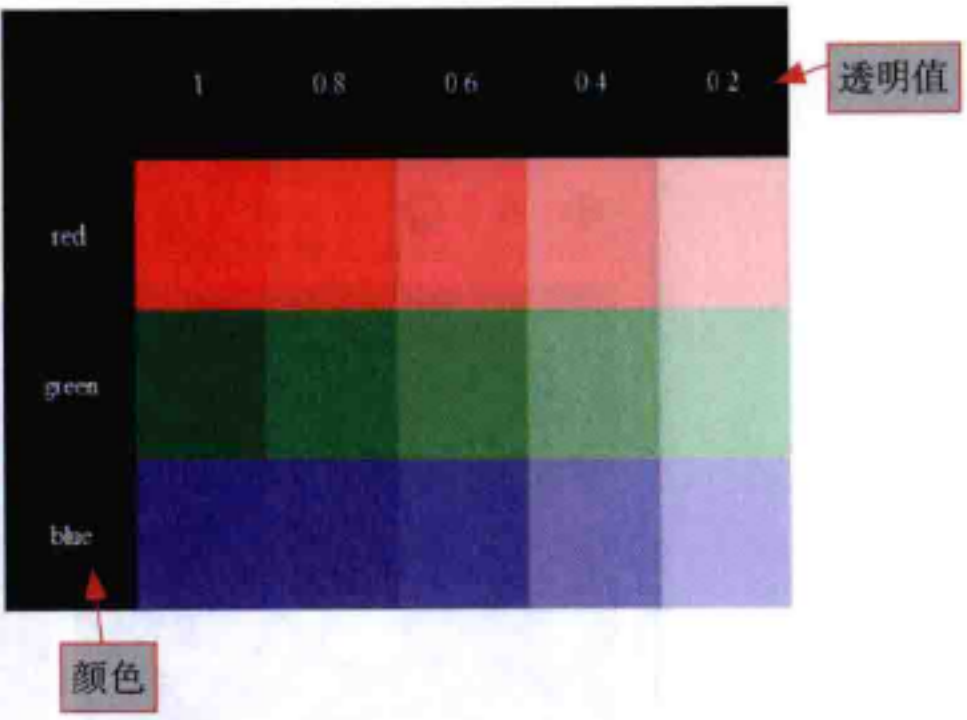


图 6-3 通过 RGBA 制作透明过渡色块

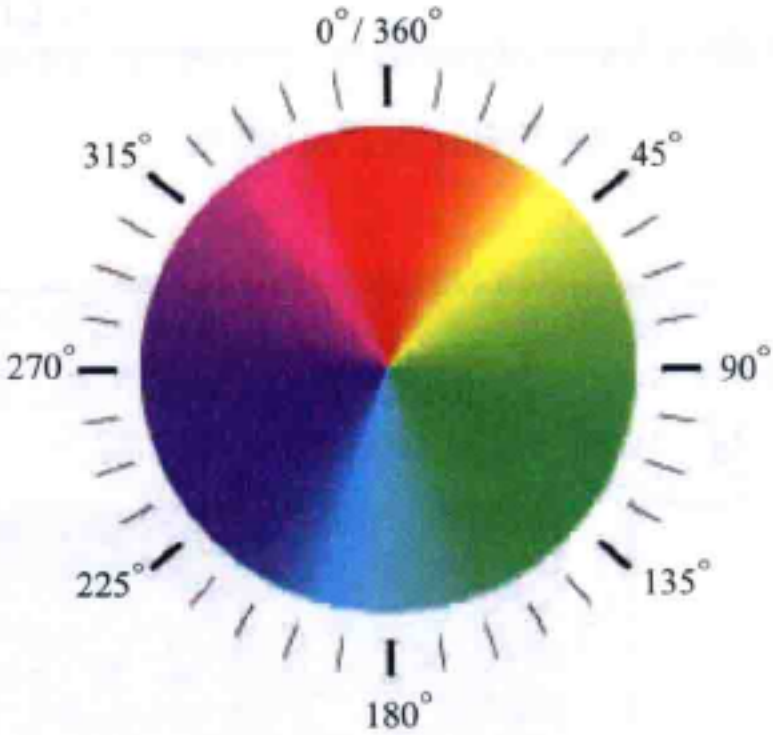


图 6-4 HSL 色盘

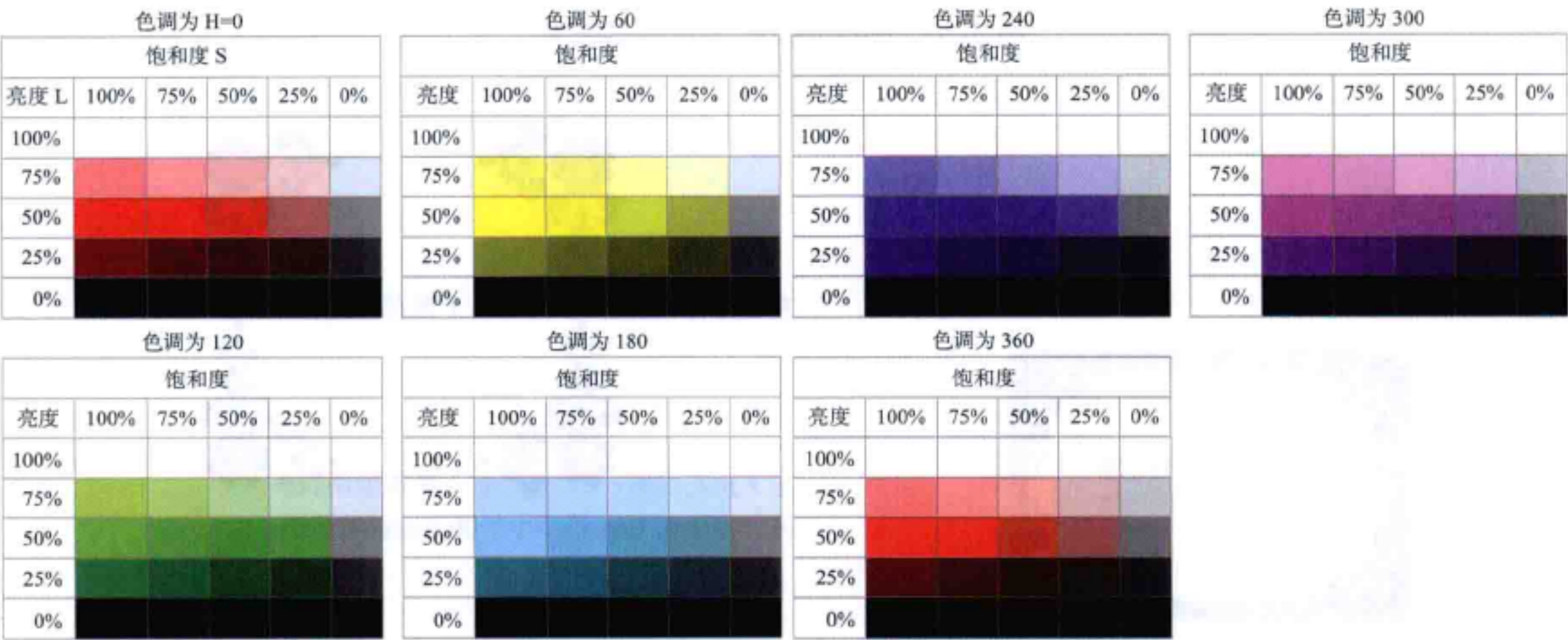


图 6-5 HSL 配色表

HSLA 制作透明过渡色块						
透明度						
色度	1	0.8	0.6	0.4	0.2	0
0						
60						
120						
180						
240						
300						
360						

图 6-6 通过 HSLA 制作的透明过渡色块

Show/Hide Extensions	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	BlackBerry Browser
								2.1	
								webkit-	
								2.2	
								webkit-	
								2.3	
								webkit-	
								3.0	
								webkit-	
								4.0	
								webkit-	
								4.1	
								webkit-	
								7.0	
								webkit-	
								4.2	
								webkit-	
								10.0	
								webkit-	
Current	10.0	20.0	-moz- 25.0	-webkit- 5.1	-webkit-	5.0-5.1	5.0-7.0		
Near future		21.0	-moz- 27.0	-webkit-	-webkit-				
Farther future		22.0	-moz- 28.0	-webkit-	-webkit-				

图 9-1 CSS3 多列布局在浏览器中的兼容性



图 10-1 渐变与渐变中的色标



图 10-5 实现顶部到底部渐变效果



图 10-6 实现底部到顶部的直线渐变效果



图 10-7 实现左向右渐变效果



图 10-8 实现右向左线性渐变效果



图 10-9 实现右下角向左上角渐变效果



图 10-10 实现左下角向右上角线性渐变



图 10-11 实现右上角向左下角线性渐变



图 10-12 实现左上角到右下角线性渐变



图 10-13 对应关键词实现的线性渐变效果

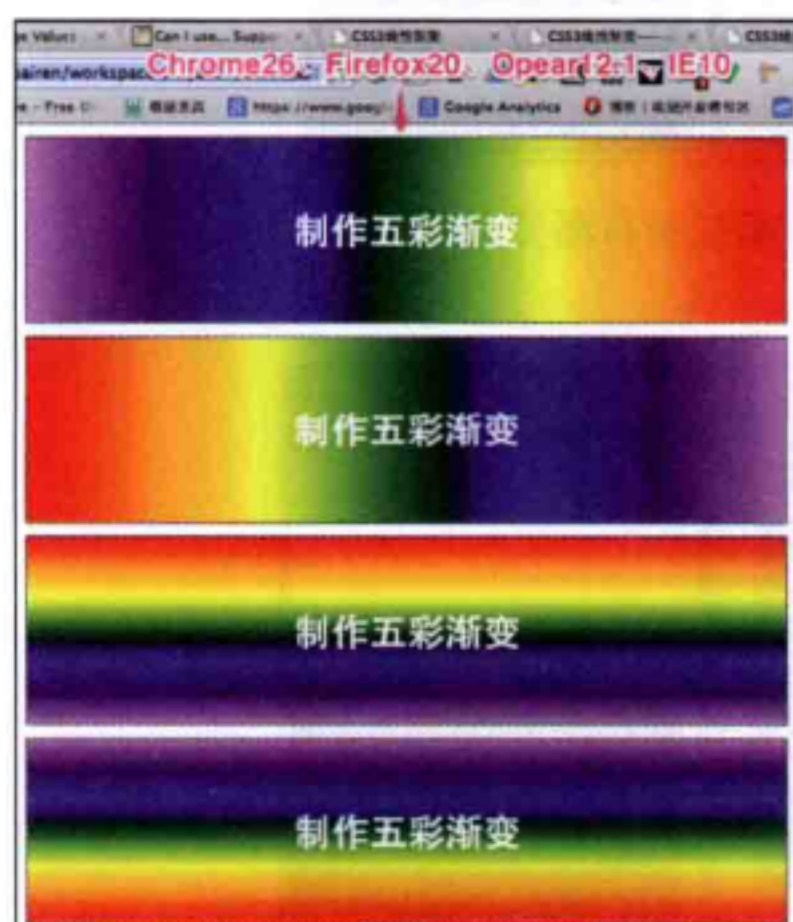


图 10-14 制作五彩渐变



图 10-15 Photoshop 中的示例线性渐变



图 10-16 自定义直线渐变

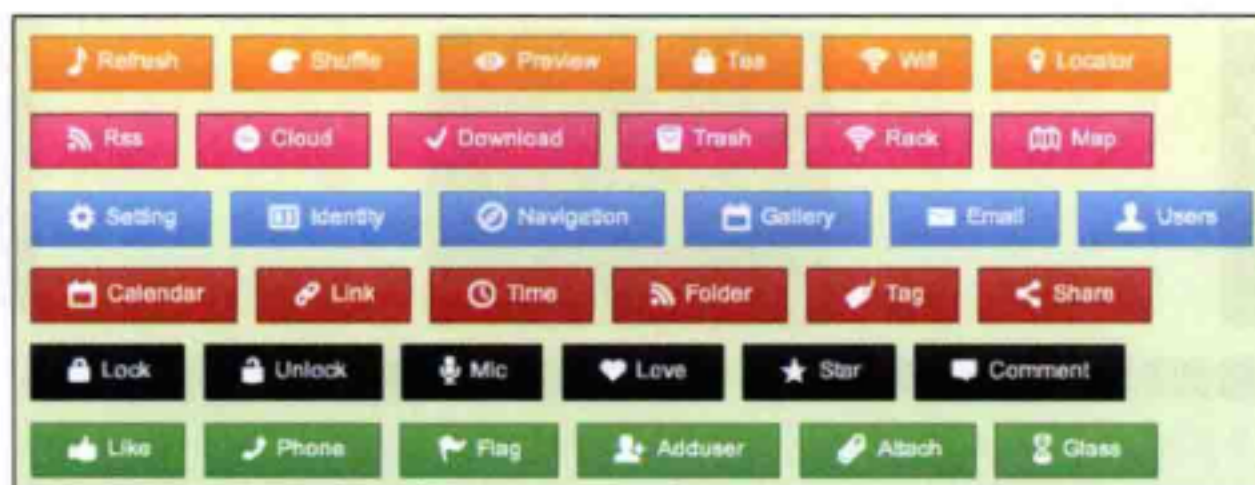


图 10-17 CSS3 渐变 linear-gradient 配合 box-shadow 实现内阴影发光按钮效果

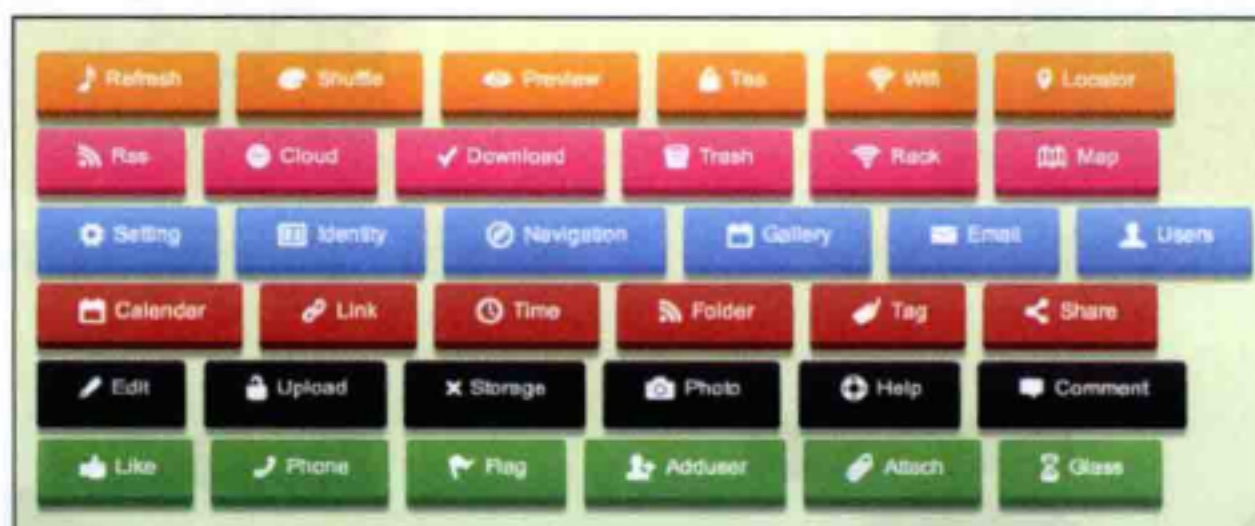


图 10-18 CSS3 渐变 linear-gradient 属性和 box-shadow 属性实现 3D 立体按钮效果

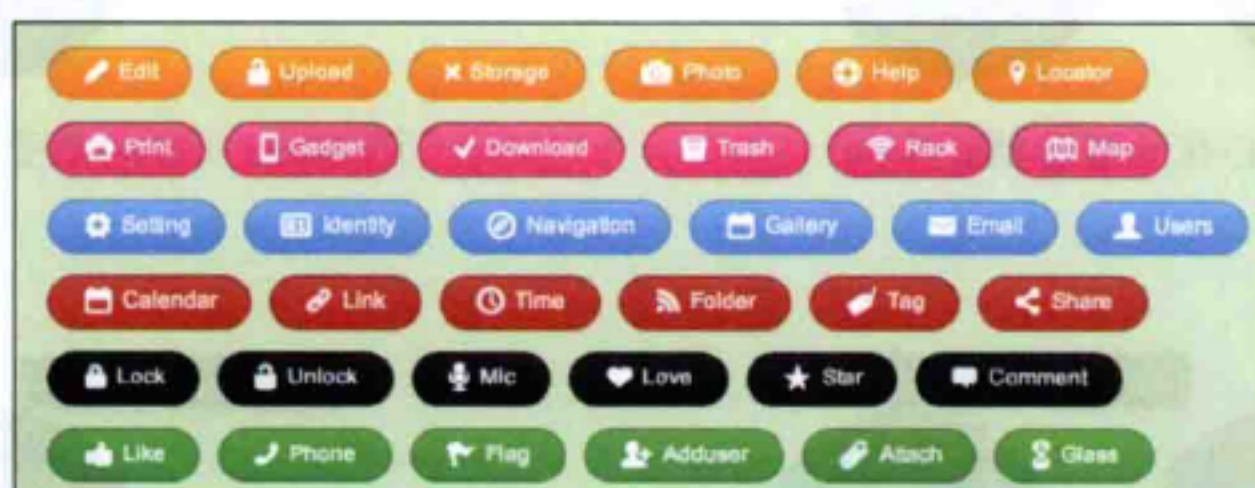


图 10-19 CSS3 渐变 linear-gradient 和 border-radius 属性实现圆角按钮效果

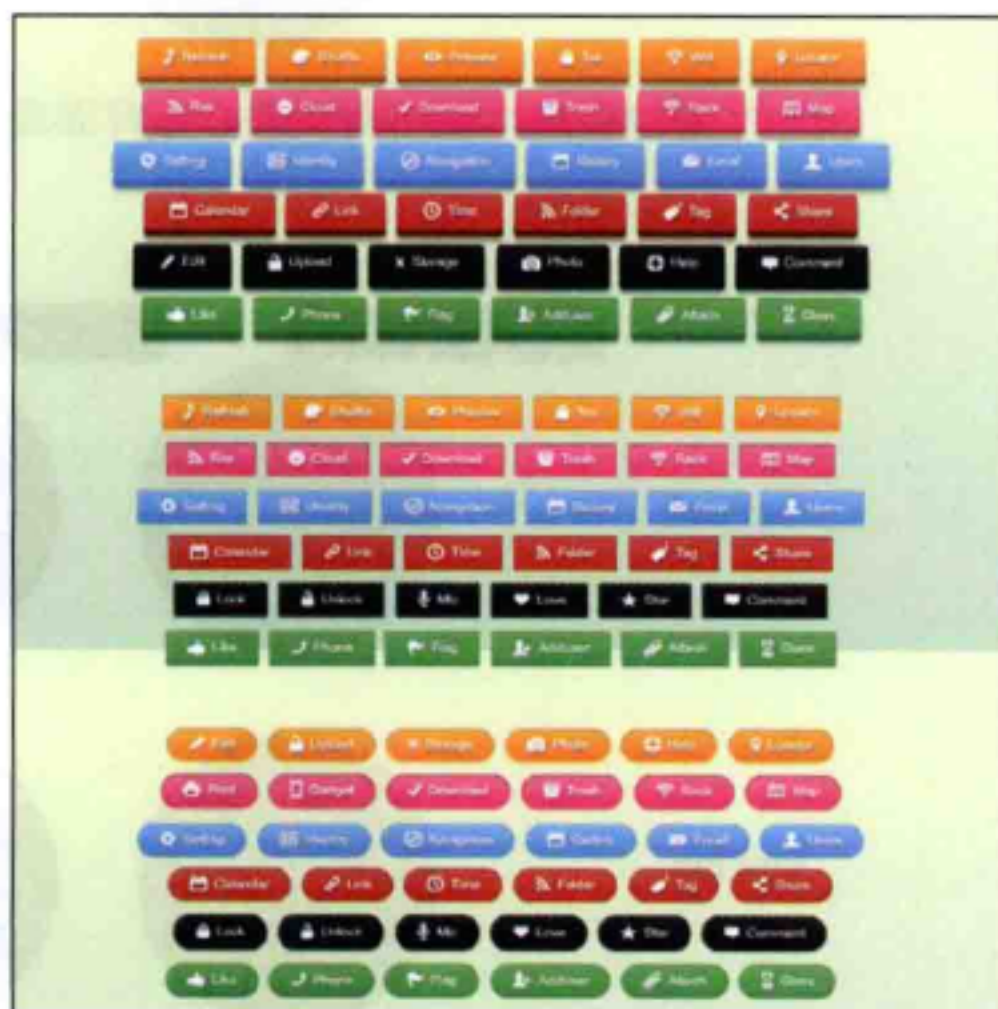


图 10-20 CSS3 的 linear-gradient、box-shadow 和 border-radius 制作按钮效果

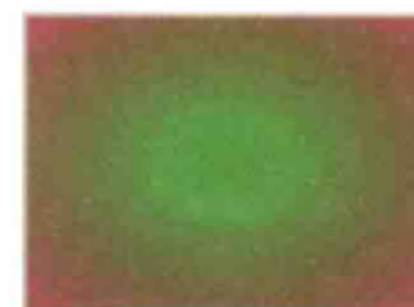


图 10-21 制作同心圆径向渐变



图 10-22 制作圆形渐变

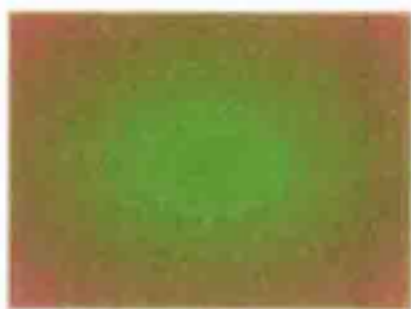


图 10-25 通过 ellipse 制作椭圆形渐变

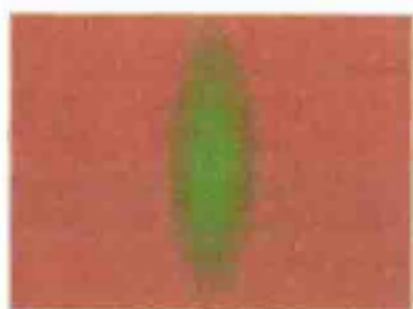


图 10-26 垂直椭圆径向渐变



图 10-27 圆心相同，内外半径相同

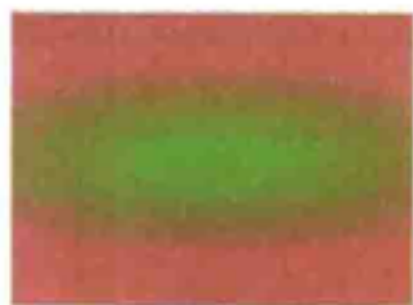


图 10-28 水平椭圆径向渐变

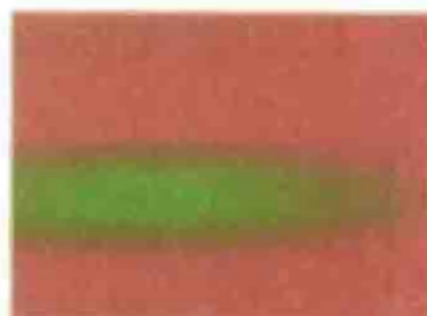


图 10-29 百分比制作椭圆形径向渐变



图 10-30 径向渐变圆心在元素容器正中间



图 10-31 径向渐变圆心在容器顶边中心点



图 10-32 径向渐变圆心在容器右边中心点



图 10-33 径向渐变圆心在容器底边中心点



图 10-34 径向渐变圆心在容器左边中心点



图 10-35 径向渐变圆心在容器左上角顶点



图 10-36 径向渐变圆心点在容器右上角顶点

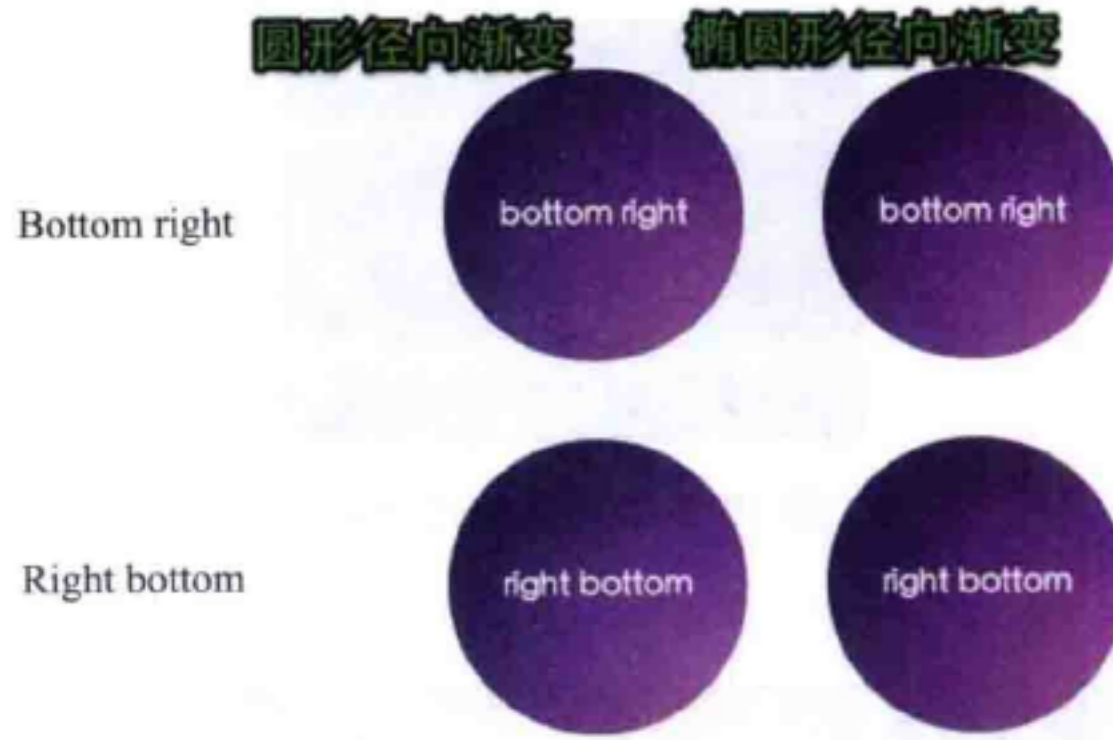


图 10-37 径向渐变圆心在容器右下角顶点

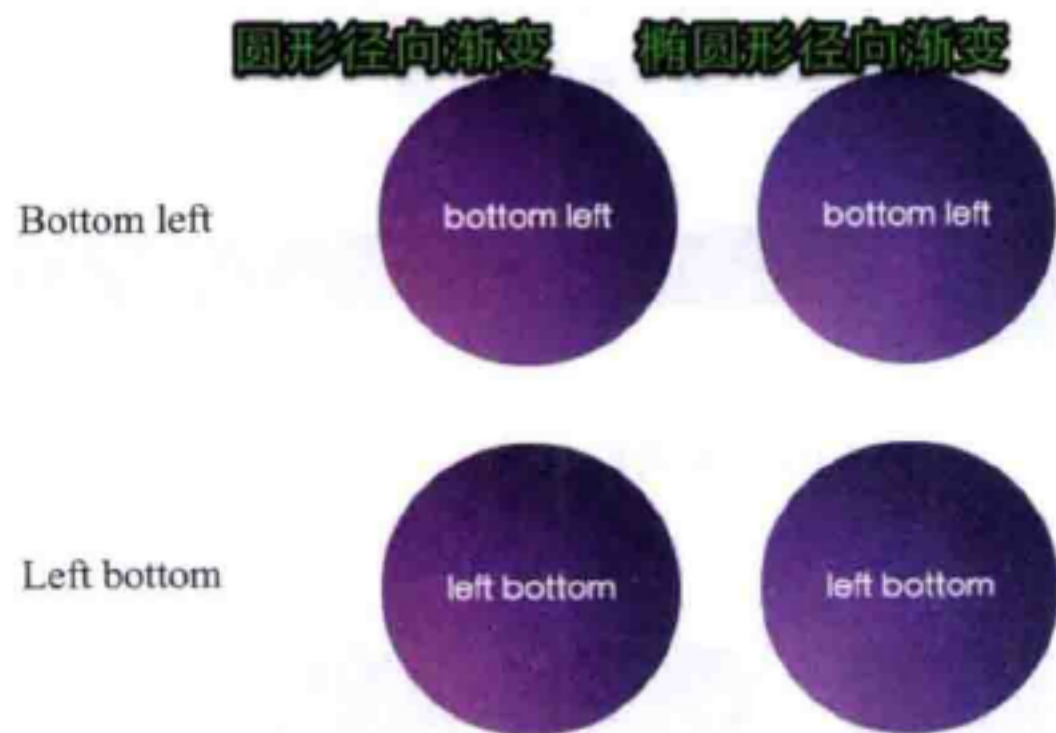


图 10-38 径向渐变圆心点在容器左下角顶点



图 10-44 三色径向渐变



图 10-45 配有具体色标位置的三色径向渐变



图 10-47 CSS3 径向渐变制作圆形图标按钮



图 10-48 斑点纹理背景



图 10-49 按钮背景



图 10-50 按钮悬浮状态

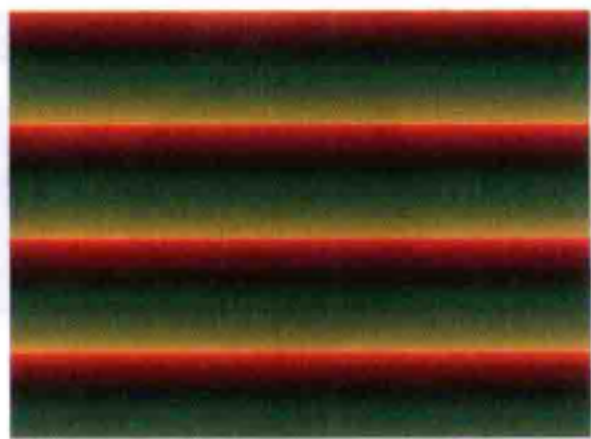


图 10-51 重复线性渐变

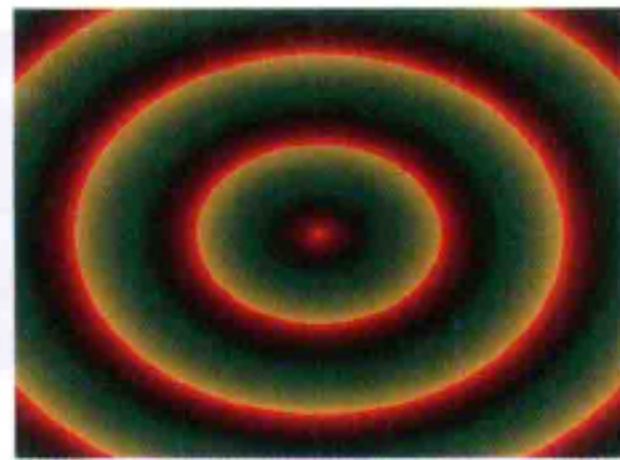


图 10-52 重复径向渐变



图 10-56 CSS3 渐变制作纹理图案

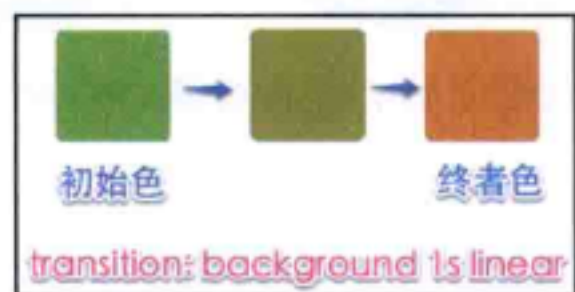


图 12-2 盒子背景色过渡



图 12-3 多属性过渡效果



图 12-4 transition-duration 效果

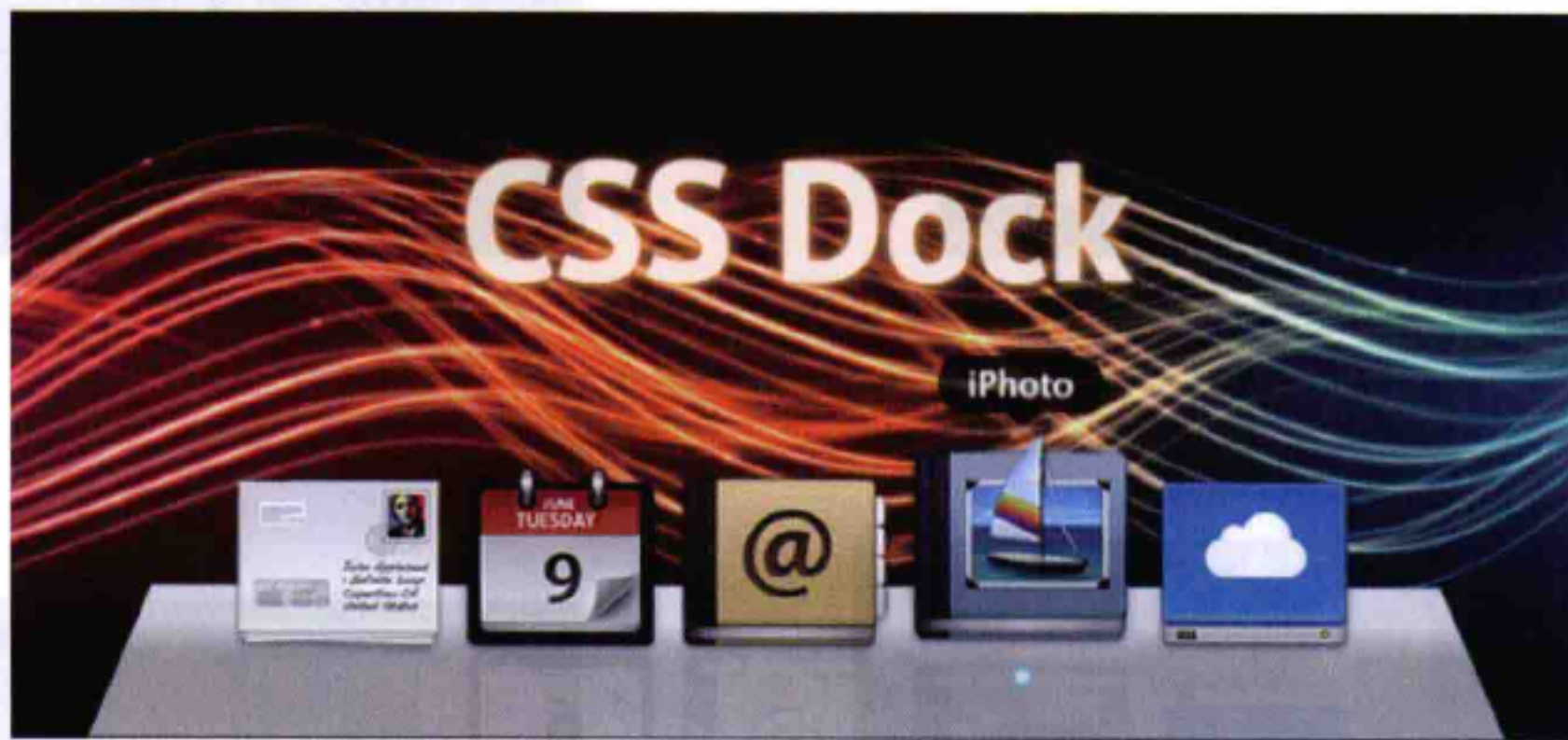


图 12-17 纯 CSS 模拟 iOS 系统桌面 Dock 导航

为什么要写这本书

CSS3 是在 CSS2.1 基础上扩展而来，事实上，它还没有完全成熟。有些专家会告诉你，CSS3 现在还用不上，甚至几年之后都不会有成熟的规范发布。

目前为止 CSS3 还没有一套成熟的规范，其中的模块也在不断更新，特别是浏览器对 CSS3 特性的支持也在不断变化，同时没有足够的时间去学习和研究 W3C 官方文档和规范，致使我们学习 CSS3 变得更为复杂。

为什么会选择这个时候编写这样一本图书呢？原因很简单。对于希望 Web 应用开发者而言，CSS3 可以说是众望所归，这也是技术变更的硬性需求。在实际 Web 应用中新标准的采纳程度正在以令人目眩的速度不断地变更着，众多浏览器厂商也在不断加快对 CSS3 新特性的支持。在编写这本图书的过程中，我也被迫不断更新书中的浏览器支持表格。

面对自己正在使用的浏览器，大多数用户并不真正了解其具备的功能有多强大。当然，他们在浏览器自动更新后可能会发现一些细微的界面变化。但他们可能不知道，新版的浏览器对哪些 CSS3 特性有所支持。

本书的目标是帮助开发者更好地掌握 CSS3 的特性，并且将新技术运用到实际的开发当中，提高自己开发 Web 程序的水平。

本书面向的读者

□ 有一定 CSS3 开发经验的前端工程师。

本书能帮助你系统掌握 CSS3 的各项知识，提升技术水平和业务能力。

□ 从事 CSS3 开发的前端工程师。

由于 CSS3 涵盖的新特性非常多，在开发过程中将本书作为速查手册，提高开发效率。

□ 前端开发爱好者。

如果还不是一名前端工程师，但是对前端开发非常感兴趣，本书也能让你对最新的 CSS 标准和规范有一个系统和全面的认识，为学习前端知识打下基础。

本书的特色

本书最大的特色就是将 CSS3 特性按模块功能分类，通过理论、图解、实战的方式向大家阐释 CSS3 每个特性功能。

□ 内容全面、丰富、翔实。

由浅到深地讲解了 CSS3 新特性的语法、特性以及使用技巧。本书涵盖了 CSS3 众多功能模块，如 CSS3 选择器特性、边框模块、文本模块、颜色模块、UI 界面模块、CSS3 动画模块、CSS 新型盒模型以及 CSS 媒体查询、响应式设计等。

□ 图解方式，直观易懂。

图解的方式是本写的最大特色之一，在描述每一个 CSS3 特性过程都配了生动的实战效果，甚至每一步骤都配有相应的效果图。就算是对文字理解或者代码理解有所误差，实战效果图能辅助你更好地理解 CSS3 每个特性。

□ 案例丰富，实战性强。

每个 CSS3 特性都配有实战体验，部分案例来自于实际开发之中。同时在每个知识点之后，还提供了综合案例。通过实践加强动手能力，更好地掌握 CSS3 中的每个知识点。

动手实践才是掌握一门新技术最有效的途径。如果能在阅读本书的过程中逐一亲手实现这些案例，那么在以后的实际开发中自然就会具有相当强的动手能力了。

本书的内容

本书包括 15 章，通过实例来演示 CSS3 模块的新特性。

第 1 章简单介绍什么是 CSS3，CSS3 的好处是什么，浏览器对 CSS3 的支持状况，以及 CSS3 带来什么新特性，并且引入渐进增强式的概念。通过对本章的学习，大家可以在一定的程度上知道一些 CSS3 的故事。

第 2 章介绍 CSS3 选择器。选择器是 CSS 中的核心部分之一，本章先阐述 CSS2 的选择器，再引入 CSS3 新增的选择器。深入介绍了 CSS3 新增选择器的功能及其实用性，还有各浏览器的兼容性。

第 3 章详细介绍 CSS3 在边框方面新增的功能特性，比如边框色、图片边框、边框圆角等，并与 CSS2 进行了对比。

第 4 章介绍 CSS3 背景功能，着重阐述了多背景、背景尺寸、背景原点方面的使用，以让

大家掌握如何使用 CSS3 背景功能的新特性。

第 5 章介绍 CSS3 文本功能。以前大家在网页制作时，只是设置文本的颜色、字体、字号等。通过对 CSS3 文本功能的学习，大家还可以运用文本阴影、文本溢出、文本换行等功能。

第 6 章介绍 CSS3 颜色特性。大家以前只有在设计软件中使用的颜色值现在都可以运用，如 RGBA、HSL、HSLA、透明度等。

第 7 章介绍 CSS3 基础盒模型与用户界面。盒模型是 CSS 的重中之重，CSS2 盒模型功能只能实现一些基本功能，对于一些特殊的功能需要借助 JavaScript 来实现。而在 CSS3 中这一点将得到很大的改善，可以通过 CSS3 来直接实现一些特殊的功能。

第 8 章介绍 CSS3 的弹性盒模型，给大家引入一种全新的布局概念，为大家的页面布局带来革命性的变化。

第 9 章介绍 CSS3 多列布局。布局在 Web 中随处可见，多列布局在 CSS2 中都是依靠 float 或者 inline-block 来实现的，而这两个属性带来的局限性也是相当大的。CSS3 多列布局将会弥补这些不足之处。

第 10 章介绍 CSS3 渐变功能。渐变效果在 Web 中也是一种常见的效果，以前靠设计师制作图片来完成，不仅增加了设计师的工作量，在页面中的效果也带来过多的局限制，扩展性也相当差。CSS3 渐变不再需要使用图片来代替这些特殊的效果。

第 11 章介绍 CSS3 变形功能。这是一个全新的功能，在 CSS2 中要实现需要借助 JavaScript。CSS3 的变形功能可以直接使用样式实现如旋转、移位、扭曲、缩放等效果。

第 12 章介绍 CSS3 过渡功能。大家在 Web 制作中，使过渡效果不再生硬，变得细腻、流畅。

第 13 章介绍 CSS3 动画功能。

第 14 章介绍 Media Query 与 Responsive 布局。随着移动设备和宽屏浏览器的普及，单一的设计不能满足 Web 页面的设计需求，此时 CSS3 的 Media Query 新特性中出现了一个新的布局概念——Responsive。本章中大家将体会到 Media Query 与 Responsive 布局的强大功能。

第 15 章介绍嵌入 Web 字体。浏览器仅限于用户在其系统上安装的字体呈现文本。CSS3 使用 @font-face 改变了这一格局。网站不再受限于少量字体，如 Arial、Verdana、Times 和 Georgia 等。

如何阅读本书

本书结构不是按层进式安排的，章节之间是按 CSS3 的模块分类，读者阅读本书时无须按照先后顺序进行，可以挑选自己喜欢的章节阅读。但如果按章节的编排顺序逐章阅读，会更

系统、更全面地学习 CSS3，从中获得最大受益。

阅读本书的案例时，尽量不要照抄书中的代码，在理解案例的设计思路基础上，自己动手开发相似功能的应用，并创造出满足自己需求的功能，举一反三。

本书中使用的约定

本书案例已在主流浏览器上进行过测试了。分别是：Firefox 12.0、Google Chrome 19.0.1084.52、Safari 5.17、Opera 11.64、IE 9。

同时在一些广泛使用的旧版本浏览器（如 IE 8）上也做了测试。很多情况下，CSS3 的效果也能体现在较低版本上，页面能保持正常阅读，而且效果也不会太差。对于每一个 CSS3 特性，将尽可能地为低版本浏览器寻求变通的备用方案，使之能兼容那些不被原生支持的浏览器。

针对每个浏览器版本，我们会标注相对应的属性在哪个版本号中开始支持。一些 CSS3 特性需要添加相应浏览器的渲染引擎的前缀才会生效，我们将会在后面的章节中依次介绍各浏览器的渲染引擎的前缀名称，以及 CSS3 特性在对应浏览器下的写法。

在阅读本书时有些约定，有必要在这里先说明。

- W3C 表示万维网联盟（World Wide Web Consortium），是制定 Web 官方标准和规范（如 CSS3）的组织。
- 初始值（即默认值）是用户不显式声明时元素所具有的属性值。需特别指明的是，属性是元素的本质，而不是用户自定义的属性。
- IE 8 及以下版本代表 IE 8、IE 7 和 IE 6。
- Webkit 引擎内核的浏览器是指 Safari（包括移动版本和桌面版本）、Google Chrome 和其他近期使用版本的 Webkit 页面渲染引擎的浏览器，其私有属性的前缀是 `-webkit-`。
- Gecko 引擎内核的浏览器是指 Mozilla，常指的是 Firefox 浏览器，其私有属性的前端缀是 `-moz-`。
- Presto 引擎内核的浏览器是指 Opera，其私有属性的前缀是 `-o-`。
- KHTML 引擎内核的浏览器是指 Konqueror，其私有属性的前缀是 `-khtml-`。
- Trident 引擎内核的浏览器是指 Internet Explorer，其私有属性的前缀是 `-ms-`。
- 在没有特别声明的情况下，本书所指的浏览器仅适用于 Windows 系统，不适用于 Mac 系统和移动端。
- 偶尔会碰到“所有浏览器”这个说法，此时仅代表目前所有广泛使用的浏览器，而非字面意义所涵盖的那些可能仅占零星市场份额的不知名的浏览器。
- “HTML”指 HTML 和 XHTML 这两种语言。

- “CSS” 指 CSS2.1 规范，除非特别声明。
- 本书所有案例代码都是以 HTML 5 的 DTD 编写。但这仅仅表示使用短小精悍的 HTML 5 文档声明 `<!DOCTYPE html>`，还有更简洁的 meta 字符编码、style 和 script 标签。没有使用任何 HTML 5 的新标签，比如 section、header、nav 和 article，所以页面可以在 IE 8 及以下版本正常运行，可以在自己的页面里将其更换为喜欢的标签。所有示例也同样兼容 HTML 4.01 和 XHTML 1.0。
- 为了方便阅读，本书中的部分案例代码仅提供了 CSS 样式代码和局部 HTML 代码，所有 CSS 实例代码必须置于一个外部样式文件或 HTML 文档的 `<head></head>` 标签内。
- 由于 CSS3 技术还在不断的完善与更新中，建议根据本书提供的参考地址，获取有关 CSS3 最新信息与更新。

勘误和支持

由于作者的水平有限，编写时间仓促，书中难免会出现一些错误或者不准确的地方，恳请读者批评指正。为此，我特意创建了一个在线支持站点 <http://www.w3cplus.com/book-comment.html>。大家可以将书中的错误发布在页面的评论中，遇到任何问题，可以留言或者发送邮件到 w3cplus@hotmail.com，我将尽量提供最满意的答案。大家还可以关注微信公众账号 ednote 进入“第三极社区”微社区与广大读者和本书作者互动。书中的全部源文件可以从华章网站（<http://www.hzbook.com>）下载，我也会将相应的功能及时更正。期待能够得到你们真挚反馈。

致谢

首先要感谢好友林小志，是他让我鼓起勇气开始写这本书，也是他一直督促我的进度，并一直鼓励我坚持到最后。同时感谢 W3CPlus (<http://www.w3cplus.com>) 社区的所有同学们一直以来对我的默默支持。

感谢机械工业出版社的编辑杨福川给我这样一个机会，在一年多的时间中始终支持我的写作，你的鼓励和帮助引导我能顺利完成全部书稿。同时也要感谢白宇编辑辛苦的付出，帮助我修改书中不足。

感谢我的爸爸、妈妈将我培养成人，并时时刻刻为我灌输着信心和力量！也要感谢我的弟弟，引导我进入这个行业，让我有机会从事喜欢的工作。感谢太太罗群英和儿子一直以来对我的支持，让我有一个安心写作的环境，并给我足够的信心去完成这本拙作。

谨以此书献给我最亲爱的家人、朋友以及众多热爱 W3CPlus 社区的朋友们！

前 言

第1章 揭开CSS3的面纱 1

- 1.1 什么是 CSS3 1
 - 1.1.1 CSS3 的新特性 2
 - 1.1.2 CSS3 的发展状况 4
 - 1.1.3 现在能使用 CSS3 吗 5
 - 1.1.4 使用 CSS3 有什么好处 5
- 1.2 浏览器对 CSS3 的支持状况 6
 - 1.2.1 经典回顾：图说浏览器大战 7
 - 1.2.2 浏览器的市场份额 8
 - 1.2.3 主流浏览器对 CSS3 支持状况 9
- 1.3 渐进增强 11
 - 1.3.1 渐进增强与优雅降级 11
 - 1.3.2 渐进增强的优点 12
- 1.4 CSS3 的现状 & 未来 13
 - 1.4.1 谁在使用 CSS3 13
 - 1.4.2 CSS3 的未来 14
- 1.5 本章小结 14

第2章 CSS3选择器 15

- 2.1 认识 CSS 选择器 15
 - 2.1.1 CSS3 选择器的优势 15

- 2.1.2 CSS3 选择器分类 16
- 2.2 基本选择器 16
 - 2.2.1 基本选择器语法 16
 - 2.2.2 浏览器兼容性 17
 - 2.2.3 实战体验：使用基本选择器 17
 - 2.2.4 通配选择器 18
 - 2.2.5 元素选择器 18
 - 2.2.6 ID 选择器 18
 - 2.2.7 类选择器 19
 - 2.2.8 群组选择器 20
- 2.3 层次选择器 21
 - 2.3.1 层次选择器语法 21
 - 2.3.2 浏览器兼容性 21
 - 2.3.3 实战体验：使用层次选择器
选择元素 21
 - 2.3.4 后代选择器 23
 - 2.3.5 子选择器 23
 - 2.3.6 相邻兄弟选择器 24
 - 2.3.7 通用兄弟选择器 25
- 2.4 动态伪类选择器 25
 - 2.4.1 动态伪类选择器语法 26
 - 2.4.2 浏览器兼容性 26
 - 2.4.3 实战体验：美化按钮 27

3.4.4 border-radius 属性的优势	115	4.4.4 实战体验: 制作全屏背景	153
3.4.5 实战体验: 制作特殊图形	115	4.5 内联元素背景图像平铺 循环方式	154
3.5 CSS3 盒子阴影属性	118	4.6 CSS3 多背景属性	154
3.5.1 box-shadow 属性的语法 及参数	118	4.6.1 CSS3 多背景语法及参数	155
3.5.2 box-shadow 属性使用方法	119	4.6.2 CSS3 多背景的优势	156
3.5.3 浏览器兼容性	129	4.6.3 浏览器兼容性	156
3.5.4 box-shadow 属性的优势	130	4.6.4 实战体验: 制作花边框	157
3.5.5 实战体验: 制作 3D 搜索 表单	130	4.7 本章小结	159
3.6 本章小结	133	第5章 CSS3文本	160
第4章 CSS3背景	134	5.1 CSS3 文本简介	160
4.1 CSS3 背景属性简介	134	5.2 CSS3 文本阴影属性	161
4.1.1 背景的基本属性	134	5.2.1 text-shadow 属性的 语法及参数	162
4.1.2 与背景相关的新增属性	137	5.2.2 浏览器兼容性	162
4.2 CSS3 背景原点属性	137	5.2.3 实战体验: 制作立体文本	163
4.2.1 background-origin 属性的 语法及参数	137	5.3 CSS3 溢出文本属性	166
4.2.2 background-origin 属性 使用方法	138	5.3.1 text-overflow 属性的语法 及参数	166
4.2.3 浏览器兼容性	140	5.3.2 浏览器兼容性	166
4.3 CSS3 背景裁切属性	141	5.3.3 text-overflow 属性使用方法	167
4.3.1 background-clip 属性的 语法及参数	141	5.3.4 实战体验: 制作固定 区域的博客列表	168
4.3.2 background-clip 属性 使用方法	143	5.4 CSS3 文本换行	170
4.3.3 浏览器兼容性	147	5.4.1 word-wrap 属性	170
4.4 CSS3 背景尺寸属性	148	5.4.2 word-break 属性	173
4.4.1 background-size 属性的 语法及参数	148	5.4.3 white-space 属性	177
4.4.2 background-size 属性使用 方法	149	5.4.4 文本换行技巧	179
4.4.3 浏览器兼容性	152	5.4.5 文本换行技术对比	180
		5.5 本章小结	180
		☆第6章 CSS3颜色特性	181
		6.1 网页中的色彩特性	181

6.1.1	网页色彩的表现原理	181	7.4	CSS3 自由缩放属性	210
6.1.2	Web 页面的安全色	182	7.4.1	resize 属性的语法及参数	210
6.1.3	色彩模式	183	7.4.2	浏览器兼容性	210
6.2	CSS3 透明属性	184	7.4.3	实战体验: 修改文本域 随意调整大小的功能	210
6.2.1	opacity 属性的语法及参数	184	7.5	CSS3 外轮廓属性	211
6.2.2	opacity 浏览器兼容性	185	7.5.1	outline 属性的语法及参数	211
6.2.3	实战体验: 制作透明 过渡色块	185	7.5.2	浏览器兼容性	212
6.3	CSS3 颜色模式	187	7.5.3	outline 和 border 的对比	212
6.3.1	RGBA 颜色模式	187	7.5.4	实战体验: 模仿边框效果	213
6.3.2	HSL 颜色模式	190	7.6	本章小结	213
6.3.3	HSLA 颜色模式	194			
6.3.4	RGBA 和 HSLA 颜色模式 之间的选择	196	第8章	CSS3伸缩布局盒模型	214
6.3.5	RGBA/HSLA 的 IE 兼容方案	196	8.1	Flexbox 模型基础知识	214
6.3.6	RGBA/HSLA 滤镜格式	197	8.1.1	CSS 中的布局模式	214
6.4	本章小结	197	8.1.2	Flexbox 模型的功能	215
			8.1.3	Flexbox 模型中的术语	215
			8.1.4	Flexbox 模型规范状态	218
			8.1.5	Flexbox 模型浏览器兼 容性	218
第7章	CSS3盒模型	198	8.1.6	Flexbox 模型语法变更	219
7.1	CSS 盒模型简介	198	8.2	旧版本 Flexbox 模型的 基本使用	221
7.1.1	什么是盒模型	198	8.2.1	伸缩容器设置 display	222
7.1.2	重置盒模型解析模式	199	8.2.2	伸缩流方向 box-orient	224
7.2	CSS3 盒模型属性	200	8.2.3	布局顺序 box-direction	226
7.2.1	box-sizing 属性的语法 及参数	200	8.2.4	伸缩换行 box-lines	229
7.2.2	浏览器兼容性	201	8.2.5	主轴对齐 box-pack	232
7.2.3	实战体验: box-sizing 拯救了布局	202	8.2.6	侧轴对齐 box-align	237
7.3	CSS3 内容溢出属性	209	8.2.7	伸缩性 box-flex	242
7.3.1	overflow-x 和 overflow-y 属性的语法及参数	209	8.2.8	显示顺序 box-ordinal-group	246
7.3.2	浏览器兼容性	209	8.2.9	实战体验: box 制作自适应的 三列等高布局	249

8.3 混合版本 Flexbox 模型的基本使用	253	9.1.2 CSS3 多列布局的属性	294
8.3.1 伸缩容器设置 display	253	9.2 CSS3 多列布局基本属性	295
8.3.2 伸缩流方向 flex-direction	254	9.2.1 columns 属性的语法及参数	295
8.3.3 伸缩换行 flex-wrap	257	9.2.2 浏览器兼容性	295
8.3.4 伸缩流方向与换行 flex-flow	259	9.2.3 实战体验: Web 页面的多列布局	296
8.3.5 主轴对齐 flex-pack	259	9.3 CSS3 多列布局列宽属性	297
8.3.6 侧轴对齐 flex-align	262	9.3.1 column-width 属性的语法及参数	297
8.3.7 堆栈伸缩行 flex-line-pack	266	9.3.2 实战体验: 浏览器根据窗口宽度变化调整列数	298
8.3.8 伸缩性 flex	271	9.4 CSS3 多列布局列数属性	302
8.3.9 显示顺序 flex-order	273	9.4.1 column-count 属性的语法及参数	302
8.4 新版本 Flexbox 模型的基本使用	275	9.4.2 实战体验: 显示固定列数	302
8.4.1 伸缩容器 display	275	9.5 CSS3 多列布局列间距属性	303
8.4.2 伸缩流方向 flex-direction	276	9.5.1 column-gap 属性的语法及参数	304
8.4.3 伸缩换行 flex-wrap	276	9.5.2 实战体验: 设置列间距	304
8.4.4 伸缩流方向与换行 flex-flow	277	9.6 CSS3 多列布局列边框样式属性	306
8.4.5 主轴对齐 justify-content	277	9.6.1 column-rule 属性的语法及参数	306
8.4.6 侧轴对齐 align-items 和 align-self	278	9.6.2 实战体验: 设置列边框	307
8.4.7 堆栈伸缩行 align-content	280	9.7 CSS3 多列布局跨列属性	309
8.4.8 伸缩性 flex	281	9.7.1 column-span 属性的语法及参数	310
8.4.9 显示顺序 order	285	9.7.2 实战体验: 文章标题跨列显示	310
8.5 综合案例: 跨浏览器的三列布局	288	9.8 CSS3 多列布局列高度属性	311
8.6 本章小结	292	9.9 本章小结	311
第9章 CSS3多列布局	293		
9.1 CSS3 多列布局简介	293		
9.1.1 浏览器兼容性	293		

☆第10章 CSS3渐变 312

- 10.1 CSS3 渐变简介 312
 - 10.1.1 什么是色标 312
 - 10.1.2 浏览器兼容性 313
- 10.2 CSS3 线性渐变 314
 - 10.2.1 CSS3 线性渐变语法与参数 315
 - 10.2.2 CSS3 线性渐变的基本用法 317
 - 10.2.3 自定义 CSS3 线性渐变 324
 - 10.2.4 实战体验: CSS3 制作渐变按钮 325
- 10.3 CSS3 径向渐变 333
 - 10.3.1 CSS3 径向渐变语法 333
 - 10.3.2 CSS3 径向渐变的属性参数 334
 - 10.3.3 CSS3 径向渐变的基本用法 335
 - 10.3.4 实战体验: CSS3 径向渐变制作圆形图标按钮 350
- 10.4 CSS3 重复渐变 353
 - 10.4.1 CSS3 重复线性渐变 353
 - 10.4.2 CSS3 重复径向渐变 354
 - 10.4.3 实战体验: 制作记事本纸张效果 354
- 10.5 综合案例: CSS3 渐变制作纹理背景 355
- 10.6 本章小结 357

第11章 CSS3变形 358

- 11.1 CSS3 变形简介 358
 - 11.1.1 CSS 变形属性及函数 358

11.1.2 浏览器兼容性 359

11.2 CSS 变形属性详解 360

- 11.2.1 transform 属性 360
- 11.2.2 transform-origin 属性 363
- 11.2.3 transform-style 属性 370
- 11.2.4 perspective 属性 372
- 11.2.5 perspective-origin 属性 377
- 11.2.6 backface-visibility 属性 380

11.3 CSS3 2D 变形 385

- 11.3.1 2D 位移 385
- 11.3.2 2D 缩放 390
- 11.3.3 2D 旋转 394
- 11.3.4 2D 倾斜 396
- 11.3.5 2D 矩阵 398

11.4 CSS3 3D 变形 403

- 11.4.1 3D 位移 404
- 11.4.2 3D 缩放 406
- 11.4.3 3D 旋转 407
- 11.4.4 3D 矩阵 409

11.5 多重变形 410

- 11.5.1 2D 多重变形制作立方体 410
- 11.5.2 3D 多重变形制作立方体 412

11.6 综合案例: 3D 变形制作产品

信息展示 413

11.7 本章小结 416

☆第12章 CSS3过渡 417

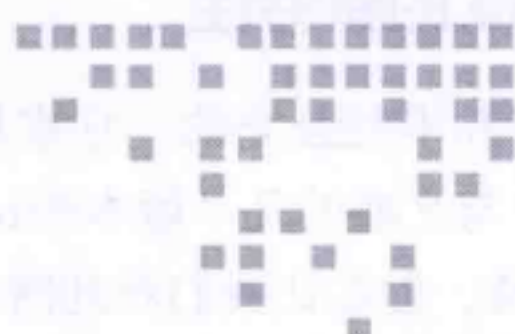
12.1 CSS3 过渡简介 417

- 12.1.1 如何创建简单的过渡 417
- 12.1.2 浏览器兼容性 418
- 12.1.3 CSS3 过渡属性 418

12.2 CSS3 过渡子属性详解 420

12.2.1 指定过渡属性	13.2.1 @keyframes 的作用
transition-property	452
12.2.2 指定过渡所需时间	13.2.2 @keyframes 的语法
transition-duration	453
12.2.3 指定过渡函数 transition-	13.2.3 浏览器兼容性
timing-function	454
12.2.4 指定过渡延迟时间	13.3 CSS 中为元素应用动画
transition-delay	454
12.2.5 多个 CSS3 过渡效果	13.3.1 使用 @keyframes 声明
433	动画
12.3 CSS3 触发过渡	454
434	13.3.2 调用 @keyframes 声明的
12.3.1 伪元素触发	动画
434	456
12.3.2 媒体查询触发	13.4 CSS3 动画子属性详解
436	457
12.3.3 JavaScript 触发	13.4.1 调用动画 animation-name
436	457
12.4 CSS3 过渡技巧	13.4.2 设置动画播放时间
437	animation-duration
12.4.1 一个完整的过渡	458
437	13.4.3 设置动画播放方式
12.4.2 可过渡的属性	animation-timing-function
438	458
12.4.3 优先的过渡属性	13.4.4 设置动画开始播放的时间
439	animation-delay
12.4.4 过渡的开始和结束为 auto	458
439	13.4.5 设置动画播放次数
12.4.5 隐式过渡	animation-iteration-count
439	458
12.4.6 开关状态的不同过渡方式	13.4.6 设置动画播放方向
440	animation-direction
12.4.7 几乎无限延迟的过渡	458
441	13.4.7 设置动画的播放状态
12.4.8 通过硬件加速过渡更加	animation-play-state
流畅	459
441	13.4.8 设置动画时间外属性
12.4.9 过渡和伪元素	animation-fill-mode
442	459
12.5 综合案例：纯 CSS3 制作 CSS	13.5 综合案例：全屏 Slideshow
Dock 导航效果	效果
443	459
12.6 本章小结	13.6 本章小结
449	464
第13章 CSS3动画	第14章 媒体特性与Responsive
450	设计
13.1 CSS3 动画简介	465
13.1.1 浏览器兼容性	14.1 媒体类型
450	465
13.1.2 CSS3 动画属性	14.1.1 Media Type 设备类型
451	465
13.2 关键帧	14.1.2 媒体类型引用方法
452	466

14.2 媒体特性	467	15.1.2 @font-face 语法	477
14.2.1 Media Query 和 CSS 属性集合	467	15.1.3 使用字体图标的优势	477
14.2.2 常用 Media Query 设备特性	468	15.2 实现 @font-face	478
14.2.3 浏览器兼容性	468	15.2.1 使用 @font-face 自定义 字体	478
14.2.4 Media Query 使用方法	468	15.2.2 声明字体来源	479
14.3 Responsive 布局概念	470	15.2.3 创建各种字体	481
14.3.1 Responsive 设计特点	471	15.2.4 调用字体	483
14.3.2 Responsive 中的术语	471	15.3 综合案例：将图标转换 成 Web 字体	483
14.3.3 Responsive 布局技巧	473	15.3.1 创建一个图标字体	483
14.3.4 meta 标签	474	15.3.2 准备插图	484
14.4 本章小结	475	15.3.3 导入到 IcoMoon	485
第15章 嵌入Web字体	476	15.3.4 从 IcoMoon 中导出字体	485
15.1 @font-face 模块介绍	476	15.3.5 下载字体文件	485
15.1.1 浏览器兼容性	476	15.3.6 调用字体	486
		15.4 本章小结	486



揭开 CSS3 的面纱

如果关注前端方面的技术，那么对 CSS 一定不会陌生，你肯定听说过 CSS3。在使用 CSS3 之前，应该对这个新一代样式表语言的来龙去脉有个基本了解。

在本章中，你将知道 CSS3 与 CSS2.1 的区别，以及当前市面上主流浏览器、移动端浏览器对 CSS3 支持的情况。对于尚不完全支持 CSS3 的浏览器，将会为大家引入一个渐进增强的概念，用一些 CSS 方法来模拟 CSS3 的实现方法。最后给大家简单介绍一些 CSS3 引入的新特性及其未来的前景。

1.1 什么是 CSS3

CSS3 并不是一门新的语言。如果接触过 CSS 就知道，CSS 是创建网页的另一个独立但并非不重要的一部分。CSS 是种层叠样式表，是一种样式语言，用来告诉浏览器如何渲染你的 Web 页面。

CSS3 是 CSS 规范的最新版本，在 CSS2.1 的基础上增加了很多强大的新功能，以帮助开发人员解决问题，并且不再需要非语义标签、复杂的 JavaScript 脚本以及图片，例如圆角功能、多背景、透明度、阴影等功能等。CSS2.1 是单一的规范，而 CSS3 被划分成几个模块组，每个模块组都有自己的规范。这样的好处是整个 CSS3 的规范发布不会因为部分难缠的部分而影响其他模块的推进。^①

现在先来看看 CSS3 激动人心的新特性。

① 更详细的信息可参见 <http://www.w3.org/Style/CSS/current-work.en.html>，其中介绍了 CSS3 具体划分为多少个模块组、CSS3 所有模块组目前所处的状态，以及将在什么时候发布。

1.1.1 CSS3 的新特性

CSS3 规范并不是独立的，它重复了 CSS 的部分内容，但在其基础上进行了很多的增补与修改。CSS3 与之前的几个版本相比，其变化是革命性的，虽然它的部分属性还不能够被浏览器完美的支持，但却让我们看到网页样式发展的前景，让我们更具有方向感、使命感。

CSS3 新特性非常多，这里挑选一些被浏览器支持较为完美、更具实用性的新特性。

1. 强大的 CSS3 选择器

使用过 jQuery 的人都知道，jQuery 的选择器功能强大，使用方便，CSS3 选择器和 jQuery 选择器非常类似。允许设计师通过选择器直接指定需要的 HTML 元素，而不需要在 HTML 中添加不必要的类名、ID 等。使用 CSS3 选择器，能在 Web 的制作中更完美地做到结构与表现分离，设计师能轻松地设计出简洁、轻量级的 Web 页面，并且能更好地维护和修改样式。

2. 抛弃图片的视觉效果

Web 中最常见的效果包括圆角、阴影、渐变背景、半透明、图片边框等。而这样的视觉效果在 CSS 中都是依赖于设计师制作图片或者 JavaScript 脚本来实现的。CSS3 的一些新特性可以用来创建一些特殊的视觉效果，后面的章节将为大家展现这些新特性是如何实现这些视觉效果。

3. 背景的转变

如果说 CSS 中的背景给你带来太多的限制，那么 CSS3 将带来革命性的变化。CSS3 不再局限于背景色、背景图像的运用，新特性中添加了多个新的属性值，例如 background-origin、background-clip、background-size，此外，还可以在一个元素上设置多个背景图片。这样，如果要设计比较复杂的 Web 页面效果，就不再需要使用一些多余标签来辅助实现了。举个例子，要实现 CSS 中的滑动门效果，在 CSS 中基本上要添加 2 ~ 3 个额外的标签来辅助实现，那么 CSS3 中的这些新特性能够在在一个标签中完成同等的效果。

4. 盒模型变化

盒模型在 CSS 中是重中之重，CSS 中的盒模型只能实现一些基本的功能，对于一些特殊的功能需要基于 JavaScript 来实现。而在 CSS3 中这一点得到了很大的改善，设计师可以直接通过 CSS3 来实现。例如，CSS3 中的弹性盒子，这个属性将给大家引入一种全新的布局概念，能轻而易举实现各种布局，特别是在移动端的布局，它的功能更是强大。大家将在第 7 章、第 8 章见识它的神功。

5. 阴影效果

阴影主要分为两种：文本阴影（text-shadow）和盒子阴影（box-shadow）。文本阴影在 CSS 中已经存在，但没有得到广泛的运用。CSS3 延续了这个特性，并进行了新的定义，该属性提供了一种新的跨浏览器方案，使文本看起来更醒目。盒子阴影的实现在 CSS 中就有点苦不堪言，为了实现这样的效果，需要新增标签、图片，而且效果还不一定完美。CSS3

的 box-shadow 将打破这种局面,可以轻易地为任何元素添加盒子阴影。

6. 多列布局与弹性盒模型布局

CSS3 引入了几个新的模块用于更方便地创建多列布局。

“多列布局”(Multi-column Layout)模块描述了如何像报纸、杂志那样,把一个简单的区块拆分成多列,第9章为大家介绍这个模块的运用。“弹性盒模型布局”(Flexible Box Layout)模块能让区块在水平、垂直方向对齐,能让区块自适应屏幕大小,相对于CSS的浮动布局、inline-block 布局、绝对定位布局来说,它显得更加方便与灵活。缺点是,这两个模块在一些浏览器中还不被支持,但随着技术的发展革新,这一刻终将到来。

7. Web 字体和 Web Font 图标

浏览器对 Web 字体有诸多限制,Web Font 图标对于设计师来说更奢侈。CSS3 重新引入 @font-face,对于设计师来说无疑是件好事。@font-face 是链接服务器上字体的一种方式,这些嵌入的字体能变成浏览器的安全字体,不再担心用户没有这些字体而无法正常显示的问题,从此告别用图片代替特殊字体的设计时代。

8. 颜色与透明度

CSS3 颜色模块的引入,实现了制作 Web 效果时不再局限于 RGB 和十六进制两种模式。CSS3 增加了 HSL、HSLA、RGBA 几种新的颜色模式。在 Web 设计中,能轻松实现某个颜色变得再亮一点或者再暗一点。其中 HSLA 和 RGBA 还增加了透明通道,能轻松地改变任何一个元素的透明度。另外,还可以使用 opacity 属性来制作元素的透明度。从此制作透明度不再依赖图片或者 JavaScript 脚本了。

9. 圆角与边框的新法

圆角是 CSS3 中使用最多的一个属性,原因很简单:圆角比直线性更美观,而且不会与设计产生任何冲突。与 CSS 制作圆角不同之处是,CSS3 无须添加任何标签元素与图片,也不需借用任何 JavaScript 脚本,一个属性就能搞定。对于边框,在 CSS 中仅局限与边框的线型、粗细、颜色的设置,如果需要特殊的边框效果,只能使用背景图片来模仿。CSS3 的 border-image 属性使元素边框的样式变得丰富起来,还可以使用该属性实现类似 background 的效果,对边框进行扭曲、拉伸和平铺等。

10. 盒容器的变形

在 CSS 时代,让某个元素变形是一个可望而不可及的想法,为了实现这样的效果,需要写大量的 JavaScript 代码。CSS3 引进了一个变形属性,可以在 2D 或者 3D 空间里操作盒容器的位置和形状,例如旋转、扭曲、缩放或者移位。我们把这些效果称为“变形”,大家将在第 11 章体验这些新特性。

11. CSS3 过渡与动画交互效果

CSS3 的“过渡”(transition)特性能在 Web 制作中实现一些简单的动画效果,让某些效果变得更具流线性、平滑性。而 CSS3 “动画”(animation)特性能够实现更复杂的样式变化,

以及一些交互效果，而不需要使用任何 Flash 或 JavaScript 脚本代码。

12. 媒体特性与 Responsive 布局

CSS3 的媒体特性可以实现一种响应式 (Responsive) 布局，使布局可以根据用户的显示终端或设备特征选择对应的样式文件，从而在不同的显示分辨率或设备下具有不同的布局渲染效果，特别是在移动端上的实现更是一种理想的做法。

1.1.2 CSS3 的发展状况

通过对 CSS3 新特性的简单介绍，大家可能要问，这些超炫的特性什么时候才能成为标准并最终发布呢？其实 CSS3 的每一个模块都有它自己的更新（进度表）时间，如图 1-1 所示^①，大家可以从这个图上看到 CSS3 的当前发展的详细进度。

Completed	Current	Upcoming	Notes	
CSS Snapshot 2010	NOTE	—	Latest stable CSS	□□
CSS Snapshot 2007	NOTE	—		□□
CSS Color Level 3	REC	REC	See Errata	□□
CSS Namespaces	REC	REC		□□
Selectors Level 3	REC	REC		□□
CSS Level 2 Revision 1	REC	REC	See Errata	□□
CSS Level 1	REC	—	Unmaintained, see Snapshot	□□
Stable	Current	Upcoming	Notes	
Media Queries	PR	REC		□□
CSS Style Attributes	CR	PR		□□
Testing	Current	Upcoming	Notes	
CSS Backgrounds and Borders Level 3	CR	PR		□□
CSS Image Values and Replaced Content Level 3	CR	PR		□□
CSS Marquee	CR	PR		□□
CSS Multi-column Layout	CR	CR		□□
CSS Speech	CR	PR		□□
CSS Mobile Profile 2.0	CR	PR	Status unknown	□□
CSS TV Profile 1.0	CR	?	Status unknown	□□
Refining	Current	Upcoming	Notes	
CSS Transforms	WD	WD		□□
CSS Transitions	WD	WD		□□
CSS Values and Units Level 3	LC	CR		□□
CSS Print Profile	LC	?	Status unknown	□□
Revising	Current	Upcoming	Notes	
CSS Animations	WD	WD		□□
CSS Flexible Box Layout	WD	WD		□□
CSS Fonts Level 3	WD	LC		□□
CSS Paged Media Level 3	LC	LC	Inactive	□□
CSS Text Level 3	WD	WD		□□
CSS Basic User Interface Level 3	CR	LC		□□
CSS Writing Modes Level 3	WD	WD		□□
CSSOM View	WD	WD		□□

图 1-1 CSS3 所有模块发展进度

① <http://www.w3.org/Style/CSS/current-work.en.html> 上的截图。还可以到地址 <http://meiert.com/en/indices/css-properties/> (CSS 各属性查询表) 查看各个 CSS 属性属于哪个 CSS 版本，以及各个属性对应的默认值，以便更清楚地知道哪些属性是在 CSS 基础上添加的。

Exploring	Current	Upcoming	Notes	
CSS Cascading and Inheritance Level 3	WD	WD	Inactive	□□
CSS Conditional Rules Level 3	WD	WD		□□
CSS Device Adaptation	WD	WD		□□
CSS Exclusions and Shapes	WD	WD		□□
CSS Generated Content for Paged Media	WD	WD		□□
CSS Grid Layout	WD	WD		□□
CSS Grid Template Layout	WD	WD		□□
CSS Line Grid	-	WD		□□
CSS Lists Level 3	WD	WD		□□
CSS Positioned Layout Level 3	WD	WD		□□
CSS Presentation Levels	WD	WD	Inactive	□□
CSS Regions	WD	WD		□□
CSS Tables Level 3	-	WD	Inactive	□□
Selectors Level 4	WD	WD		□□
CSS Object Model	WD	WD		□□
CSS Fragmentation Level 3	WD	WD		□□
Compositing and Blending	-	WD		□□
Filter Effects	-	WD		□□
Rewriting	Current	Upcoming	Notes	
CSS Basic Box Model Level 3	WD	WD	Dangerously outdated; see CSS2.1.	□□
CSS Generated Content Level 3	WD	WD	Severely outdated	□□
CSS Line Layout Level 3	WD	WD	Severely outdated	□□
CSS Ruby	WD	WD	Outdated and majorly underdefined	□□
CSS Syntax Level 3	WD	WD	Severely outdated; see CSS2.1.	□□
Abandoned	Current	Upcoming	Notes	
Behavioral Extensions to CSS	WD	-		□□
CSS Hyperlink Presentation	WD	-		□□
CSS Grid Positioning	WD	-		□□

WD

 工作草案

LC

 最终发布

CR

 候选推荐标准

PR

 提名推荐标准

REC

 推荐标准

图 1-1 (续)

Web 开发者希望在 CSS3 标准规范发布之前就能使用这些新特性，而它们的使用还受限于不同的浏览器，只有浏览器完全支持了，才能完全使用这些新特性。

目前，CSS3 还不是最终的标准，有很多浏览器支持不够完美，那么现在可以使用 CSS3 吗？

1.1.3 现在能使用 CSS3 吗

从图 1-1 中可以看出，CSS3 还在不断完善中，很多功能还处于草稿阶段，但部分模块进入了“候选推荐”状态，这说明在 Web 设计中完全可以使用这些模块。即使有一些模块还处于“工作草案状态”，也可以尝试着使用，只有不断将新的 CSS 技术运用到实际工作中，才能发现应用这些新技术所面临的真正挑战，以便 W3C 更好地完善它们，从而更好地、有效地促使它们成为真正的标准。

你应该了解哪些可用，哪些还不能使用。换句话说，在实际工作开发中可以先运用相对稳定的 CSS3 特性，并确保不会对尚不支持这些特性的浏览器造成影响。做到明智的使用，而不是盲目地滥用 CSS3 新特性。

1.1.4 使用 CSS3 有什么好处

与 CSS 相比，使用 CSS3 有什么好处呢？最明显的就是 CSS3 能让页面看起来非常炫、

非常酷，使网站设计锦上添花，但它的好处远远不止这些。在大多数情况下，使用 CSS3 不仅有利于开发与维护，还能提高网站的性能。与此同时，还能增加网站的可访问性、可用性，使网站能适配更多的设备，甚至还可以优化网站 SEO，提高网站的搜索排名结果。下面介绍 CSS3 特有的好处。

1. 减少开发与维护成本

为什么说 CSS3 能减少开发与维护的成本呢？先来看一个实例。一个圆角效果，在 CSS 中需要添加额外的 HTML 标签，使用一个或者更多图片来完成，而使用 CSS3 只需要一个标签、一个 border-radius 属性就能完成。这样，CSS3 技术能把从绘图、切图和优化图片的工作中解救出来。

如果后续需要调整这个圆角的弧度或者圆角的颜色，使用 CSS，要从头绘图、切图才能完成，而使用 CSS3 几秒就完成这些工作。

CSS3 还能使你远离一大堆的 JavaScript 脚本代码或者 Flash，你不再需要花大把时间去写脚本或者寻找合适的脚本插件并修改以适配你的网站特效。

最后，有些 CSS3 技术还能帮你进行“减肥”，让结构更加“苗条”。你不用为了达到一个效果而嵌套很多 DIV 和类名，这样能有效地提高工作效率、减少开发时间、降低开发成本。例如，制作一个重叠的背景效果，在 CSS 中需要添加 DIV 标签和类名，在不同的 DIV 中放一张背景图，现在可以使用 CSS3 的多背景和背景尺寸等新特性，在一个 DIV 标签中完成这些工作。

2. 提高页面性能

很多 CSS3 技术通过提供相同的视觉效果而成为图片的“替代品”，换句话说，在进行 Web 开发时，减少多余的标签嵌套，以及图片的使用数量，意味着用户要下载的内容将会更少，页面加载也会更快。另外，更少的图片、脚本和 Flash 文件让 Web 站点减少 HTTP 请求数，这是提升页面加载速度的最佳方法之一。而使用 CSS3 制作图形化网站无需任何图片，极大地减少 HTTP 的请求数量，并且提升页面的加载速度。当然，这取决于采用 CSS3 特性来代替什么技术，同样还要看如何使用 CSS3 特性。例如 CSS3 的动画效果，能够减少对 JavaScript 和 Flash 文件的 HTTP 请求，但可能要求浏览器执行很多的工作来完成这个动画效果的渲染，这有可能导致浏览器响应缓慢，致使用户流失。因此，在使用一些复杂的特效时，大家需要考虑清楚。不过这样的现象毕竟为数不多。其实很多 CSS3 技术能够在任何情况下都大幅提高页面的性能。就这一条足以让我们使用 CSS3。

1.2 浏览器对 CSS3 的支持状况

CSS3 给我们带来众多全新的设计体验，但有一个问题值得考虑——浏览器对 CSS3 特

性的支持情况如何？因为页面最终离不开浏览器来渲染，并不是所有浏览器都完全支持 CSS3 的特性。有时候花时间写的效果只能在特定的浏览器下有效，意味着只有部分用户能欣赏到，这样的工作变得没有什么意义。例如，使用 CSS3 制作背景仅在 WebKit 内核的浏览器下有效果。

想知道用户能够体验到哪些 CSS3 的新特性，必须了解当前浏览器对 CSS3 特性的支持程度如何。

1.2.1 经典回顾：图说浏览器大战

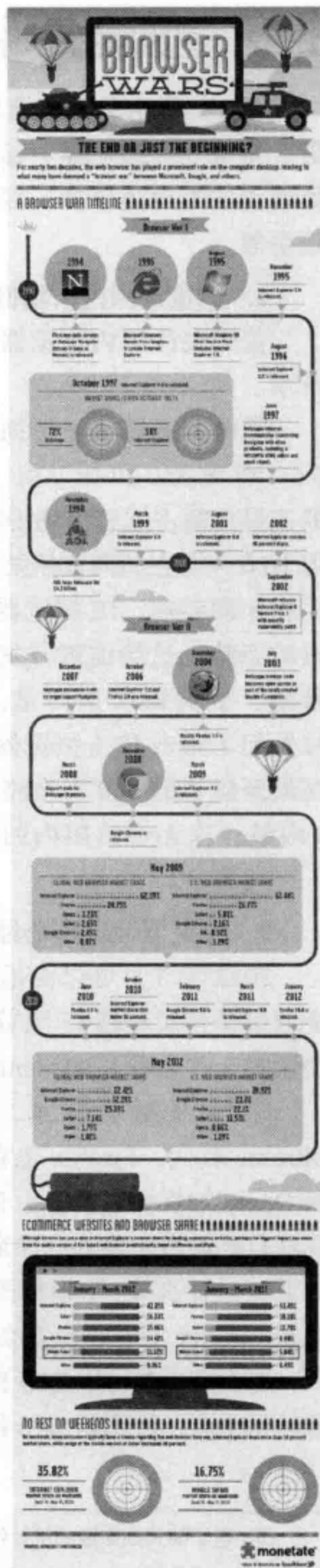
“浏览器大战”一词在 20 世纪末产生，网景（Netscape）与微软展开了第一次互联网大战，结果是网景以失败告终，微软荣登冠军宝座。

2004 年 11 月 Firefox 1.0 诞生，浏览器开始了历史上的第二次大战，IE 的地位受到了以 Firefox 为首的其他浏览器的挑战。2008 年 12 月 Google Chrome 的诞生，向市场投放了一颗重磅炸弹。此时的 IE 也开始了版本的升级，虽然 IE 将版本更新到了 IE 8，但面对 Firefox 和 Google Chrome 两个强劲的对手，其更新的步伐依然显得太慢，在 2010 年 IE 的市场份额跌至 50%。而后，Chrome 不断更新，其市场份额快速上升。2012 年 5 月，终于夺得浏览器的霸主之位。

这不是浏览器大战的结束，仅仅是 IE 时代在落幕而已。随着移动设备的风靡，移动版本 Safari 的市场份额在一年的时间里迅速增长。也许，第三次浏览器大战的战场并不在桌面领域，而是在移动领域。

市面上浏览器品种繁多，从而引发浏览器的市场大战，这场战争持续了近二十年，但从未有结束战争的趋势。浏览器之争提示了 Web 浏览器的影响，比如 Chrome 和 Firefox 对浏览器霸主 IE 发起的挑战，随着移动终端的出现，另一个强有力的竞争者——移动 Safari 网络浏览器也加入这场无休止的浏览器之战。下面一起来看看 monetate.com^①绘制的浏览器战争图，如图 1-2 所示。

① 详情见 <http://monetate.com/infographic/browser-wars-the-end-or-just-the-beginning>。



1.2.2 浏览器的市场份额

图 1-2 所示只是主流浏览器的市场之争，国内还有许多国产的浏览器，例如 QQ 浏览器、奇虎 360 浏览器、移动端的 UC 浏览器等。用户在使用什么样的浏览器，这个使用率始终无法准确地掌握，因为这个概率始终都在变化，下面详细看看国内和全球浏览器的市场份额。

1. 浏览器国内市场份额

首先关注国内浏览器的市场份额，一起来看百度统计的浏览器市场份额图，如图 1-3 所示。

国内浏览器市场位列三甲的分别是 IE 6.0、奇虎 360 和 IE 7.0，三个版本的浏览器流量份额占据总市场份额的 61.1% 左右，但 IE 6 ~ 8 在国内依然处于绝对领先态势，但相比两年前，IE 浏览器在国内也步入下滑的态势，这给前端开发人员带来一丝的希望。更值得庆幸的是，360 浏览器在 5 月发起了狙击 IE 6 浏览器的活动，并开始 in 最新版本的 360 安全浏览器中内置了 IE 8 内核，这无疑给国内的前端工程师带来了一丝清凉。

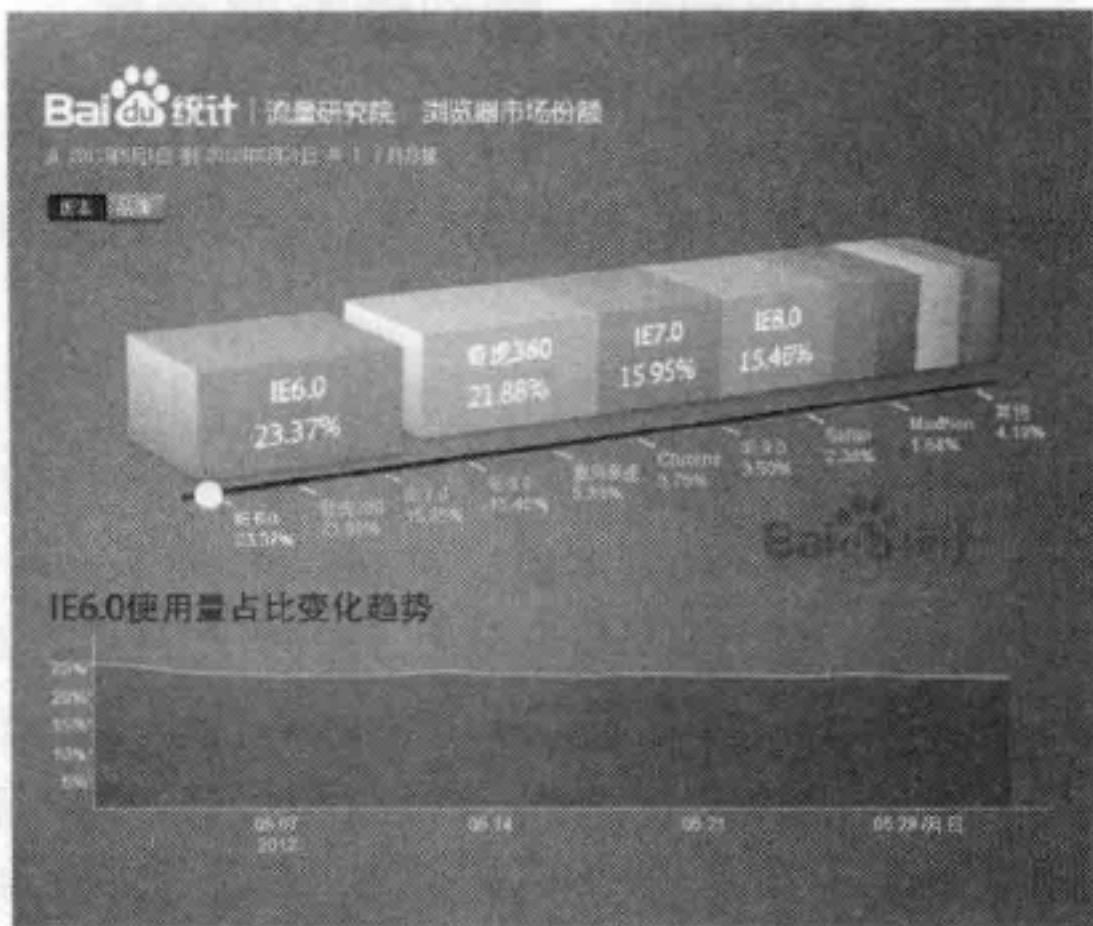


图 1-3 国内浏览器市场份额

2. 浏览器全球市场份额

2012 年 5 月可以说是浏览器厂商激烈竞争的一个月，一度报出 Google Chrome 浏览器全球份额首次超越 IE 浏览器，夺得浏览器全球霸主之位。全球浏览器市场份额发生了哪些变化呢？首先看 StatCounter 的统计数据^①，如图 1-4 所示。

图 1-4 中数据显示，在 2012 年 5 月，IE 浏览器已失去了浏览器的霸主之位，被 Chrome 取代，Firefox 也在市场上位居第三。如果将其他版本的 Firefox、Chrome、Safari 和 Opera 加在一起计算，IE 所占的市场份额确实已少于这些符合标准的现代浏览器。通常，我在自己站点上发布一个新的 CSS3 技巧时，很多朋友会问：“它在 IE 浏览器上能运行吗？效果又是什么样？”根据图 1-4 的显示结构，是不是应该换一种思维，是不是询问“这个效果在 Firefox 上看起来怎么样”更有意义呢？

那么是不是可以忽略 IE 呢？其实不然，IE 虽然在全球市场份额不再是霸主，但在国内它依然是主流，特别是 IE 6 依然占有半壁江山，这也致使我们不能不考虑使用 IE 的用户群。

① 想了解其他来源的统计结果，可以访问维基百科的“Usage share of web browsers”页面 (http://en.wikipedia.org/wiki/Usage_share_of_web_browsers)。

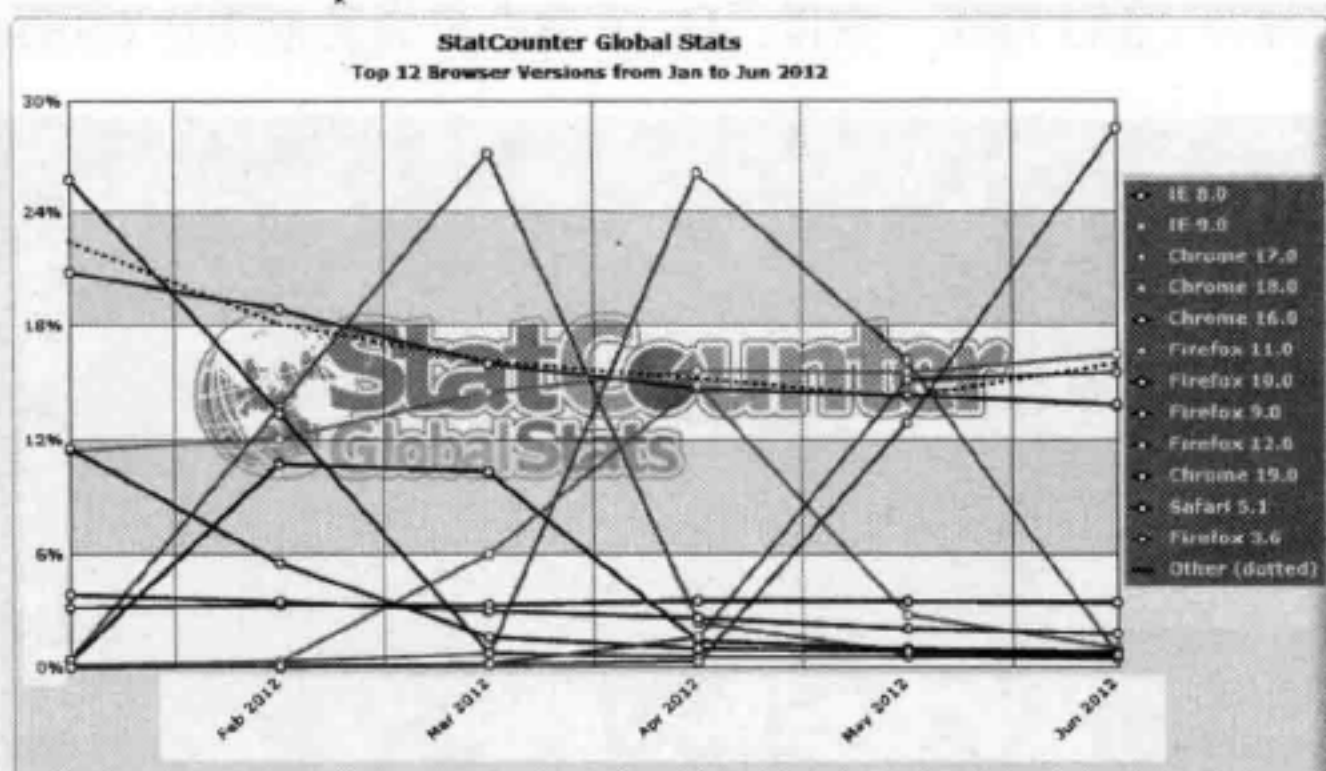


图 1-4 主流浏览器市场份额（2012 年 1 月～2012 年 6 月）

制作一个网站，其内容应该在任何浏览器上都是可用的，不应该忽略或抛弃某些用户。虽然制作一个令人满意的 Web 页面不是一件难事，但是为了一个渐渐消失的用户群体花费大量的时间与成本确实不是一个明智之举。

正如前面所讲，CSS3 对网站意义何在，应该是由用户群体来确定，而不是由浏览器的市场份额所决定。换言之，除非网站统计结果与这个结果有很大的出入，否则就不应该继续认为非 IE 用户仅仅是个不需要特别关注的边缘化群体。在非 IE 浏览器与 IE 浏览器上花费的时间同样重要。而 CSS3 能很容易地让网站在非 IE 浏览器下更棒，而且少数情况之下这些 CSS3 属性也适合 IE 浏览器。

1.2.3 主流浏览器对 CSS3 支持状况

幸运的是，CSS3 特性大部分都已经有了很好的浏览器支持度（后面在讲每个 CSS3 特性之时，会列出各浏览器对其支持情况）。各大主流浏览器对 CSS3 的支持越来越完善，曾经让多少前端开发人员心碎的 IE 也开始挺进 CSS3 标准行列。当然，即使 CSS3 标准制定完成，现代浏览器要普及到大部分用户也是一个相当漫长的过程。如果现在就要使用 CSS3 来美化你的站点，有必要对各大主流浏览器对其新技术的支持情况有一个全面的了解。

本节分别在 Mac 和 Windows 两个平台介绍 Chrome、Firefox、Safari、Opera 和 IE 五大主流浏览器对 CSS3 新特性和 CSS3 选择器的支持情况。

1. 主流浏览器对 CSS3 属性的支持状况

图 1-5 所示是 findmebyIP^① 为五大主流浏览器对 CSS3 属性支持状况的统计图。

从图中可以看出，CSS3 中的 Overflow Scrolling 属性还没有浏览器支持，其他属性在 Mac 系统下，Chrome、Safari 都支持，其次支持较好的是 Firefox 和 Opera。而在 Windows 系统下，WebKit 内核浏览器表现的非常优秀，其次是 Firefox 和 Opera。同时 IE 也迎头追

① 详情见 <http://fmbip.com/litmus>。

赶，在 IE 9 中支持部分 CSS3 特性。据说，IE 10 将会更完美地支持 CSS3。










	MAC					WIN								
														
	SAFARI	FIREFOX	OPERA	CHROME	SAFARI	IE				FIREFOX	OPERA	CHROME		
	5.1	11	11.62	18	5.1	6	7	8	9	11	11.61	18		
RGBA	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	89%	
HSLA	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	89%	
Box Sizing	✓	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓		
Background Size	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	87%	
Multiple Backgrounds	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	87%	
Border Image	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓	85%	
Border Radius	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	89%	
Box Shadow	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	88%	
Text Shadow	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓	69%	
Opacity	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	89%	
CSS Animations	✓	✓	✗	✓	✓	✗	✗	✗	✗	✓	✗	✓	56%	
CSS Columns	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓	84%	
CSS Gradients	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓	82%	
CSS Reflections	✓	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗	✓	43%	
CSS Transforms	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	88%	
CSS Transforms 3D	✓	✓	✗	✓	✓	✗	✗	✗	✗	✓	✗	✓	23%	
CSS Transitions	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓	64%	
CSS FontFace	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	97%	
FlexBox	✓	✓	✗	✓	✓	✗	✗	✗	✗	✓	✗	✓	66%	
Generated Content	✓	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓		
DataURI	✓	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓		
Pointer Events	✓	✓	✗	✓	✓	✗	✗	✗	✗	✓	✗	✓		
Display: table	✓	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓		
Overflow Scrolling	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗		
Media Queries	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓		

图 1-5 主流浏览器对 CSS3 属性支持状况

2. 主流浏览器对 CSS3 选择器的支持状况

图 1-6 所示是 findmebyIP^①为五大主流浏览器对 CSS3 选择器的支持状况统计图。值得高兴的是，图中除了 IE 6～8 有“×”之外，其他浏览器都是“√”，全部支持 CSS3 选择器。

从图 1-5 和图 1-6 中可以清楚地知道，无论是在 Mac 系统下还是在 Windows 系统下，Google Chrome 和 Safari 对 CSS3 特性的支持度是最好的，而 IE 系列是最差的，特别是 IE 6～8。原因很简单，IE 6～8 发布于 CSS3 完善之前。

差别各异的浏览器致使页面在不同的浏览器之中渲染并不一致。特别是在当今这个信息发达的时代，设备、屏幕、浏览器的形态越来越丰富，人们的习惯设置也不尽相同，因此想再创造一个在任何地方都表现一致的页面就更加的不可能。只要你关注如何提供实用、易用、好用的页面，这点表面上的差异就显得不重要。而这种想法就是接下来要介绍的“渐进增强”。

① 详情见 <http://fmbip.com/litmus>。














	MAC					WIN								
														
	SAFARI	FIREFOX	OPERA	CHROME	SAFARI	IE	FIREFOX	OPERA	CHROME	SAFARI	FIREFOX	OPERA	CHROME	
	5.1	11	11.62	18	5.1	6	7	8	9	11	11.61	11.61	18	
Begins with	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓	✓	98%
Ends with	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓	✓	98%
Matches	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓	✓	98%
Root	✓	✓	✓	✓	✓	×	×	×	✓	✓	✓	✓	✓	89%
nth-child	✓	✓	✓	✓	✓	×	×	×	✓	✓	✓	✓	✓	89%
nth-last-child	✓	✓	✓	✓	✓	×	×	×	✓	✓	✓	✓	✓	89%
nth-of-type	✓	✓	✓	✓	✓	×	×	×	✓	✓	✓	✓	✓	89%
nth-last-of-type	✓	✓	✓	✓	✓	×	×	×	✓	✓	✓	✓	✓	89%
last-child	✓	✓	✓	✓	✓	×	×	×	✓	✓	✓	✓	✓	89%
first-of-type	✓	✓	✓	✓	✓	×	×	×	✓	✓	✓	✓	✓	89%
last-of-type	✓	✓	✓	✓	✓	×	×	×	✓	✓	✓	✓	✓	89%
only-child	✓	✓	✓	✓	✓	×	×	×	✓	✓	✓	✓	✓	89%
only-of-type	✓	✓	✓	✓	✓	×	×	×	✓	✓	✓	✓	✓	89%
empty	✓	✓	✓	✓	✓	×	×	×	✓	✓	✓	✓	✓	89%
target	✓	✓	✓	✓	✓	×	×	×	✓	✓	✓	✓	✓	88%
enabled	✓	✓	✓	✓	✓	×	×	×	✓	✓	✓	✓	✓	89%
disabled	✓	✓	✓	✓	✓	×	×	×	✓	✓	✓	✓	✓	89%
checked	✓	✓	✓	✓	✓	×	×	×	✓	✓	✓	✓	✓	89%
not	✓	✓	✓	✓	✓	×	×	×	✓	✓	✓	✓	✓	89%
General Sibling	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓	✓	98%

图 1-6 主流浏览器对 CSS3 选择器的支持状况

1.3 渐进增强

第一次听到“渐进增强”(Progressive Enhancement)一词是在前端重构交流会上。渐进增强并不是一种技术，而是一种开发的方式，更是一种 Web 设计理念。首先思考一个问题：“网站是否需要在每个浏览器中看起来都一样？”带着这个问题来看渐进增强。

1.3.1 渐进增强与优雅降级

正如前面所言，渐进增强是一种开发方式，更是一种设计理念。在编写 Web 页面时，首先保证最核心的功能实现，让任何低端的浏览器都能看到站点内容，然后考虑使用高级但非必要的 CSS 和 JavaScript 等增强功能，为当前和未来的浏览器提供更好的支持，给用户带来更好的体验。

在设计的时候，先考虑低端设备用户能否看到所有内容，然后在此基础上为高端用户进行设计。不仅为高端设备用户提供一个完美的应用，也要为不同性能级别设备的用户设计不同级别的不那么完美的应用。这称为“优雅降级”。

目前而言，虽对“渐进增强”有所了解的人很多，但是要说普及或深入还远远没到時候。在大家平时的设计思维中有一种极强的固定思维，也就是想让网站在各个浏览器下表现一致。这种出发点本身并没有什么問題，但是这样会让领先的浏览器的优势无法充分显示出来。

因此，从今天开始要改变制作 Web 站点的思维，让网站能优雅降级，目标应该是——向尽可能多的用户提供尽可能优质的用户体验。这跟用户访问网站使用方式无关，无论通过 iPhone、高端的桌面系统、Kindle，还是阅读器，用户都能得到尽可能独特且完美的体验。

1.3.2 渐进增强的优点

“向尽可能多的用户提供尽可能优质的用户体验”这一目标听起来相当不错。有的人制作 Web 站点时报怨，IE 怎样才具有 CSS3 的效果。诚然，我们不使用渐进增强也可以实现，如为一些旧浏览器提供一套兼容方案，确保页面与现代浏览器保持一致。简单来说就是在支持 CSS3 的浏览器中使用 CSS3 特性，在不支持的浏览器中另写一套样式来模拟 CSS3 效果，实现让网站在所有浏览器都一样。

可以说，通过这种方法只是让低版本的浏览器渲染页面更好看一点，并没有得到实质性的提高。

因此，如果网站始终无法做到一模一样，为什么不使用 CSS3 技术使它在现代浏览器上看起来更靓丽呢？当然，某些 CSS3 特性在不支持的浏览器中是“无法模拟”的，但使用渐进增强，就无须为了让网站适合所有人而放弃这些技术。

CSS 的渐进增强有别于 CSS 的 hack。hack 是浏览器厂商的一种手法，用来增强自己的竞争，而渐进增强起到锦上添花之效。所以前者应该尽量避免使用，而后者应该适当使用。

就样式而言，渐进增强的对象是一些现代浏览器，渐进增强的一些属性主要是 CSS3 的一些特性，或是 IE 低级版本不支持的一些属性，或是其他一些特殊情况。淘宝网上的一个例子如图 1-7 所示。

这里采用了 CSS3 的旋转特性 transform，鼠标移上去时，三角会实现旋转的动画效果，并改变方向。之前，要实现图 1-7 所示的旋转效果，需要一大串 JavaScript 脚本。而使用 transform 仅仅需要几行代码。

```
#site-nave .menu: hover .menu-hd {
    -webkit-transform: rotate(180deg);
    -webkit-transform-origin-x: 50%;
    -webkit-transform-origin-y: 30%;
    -moz-transform: rotate(180deg);
    -moz-transform-origin-x: 50%;
    -moz-transform-origin-y: 30%;
    -o-transform: rotate(180deg);
    -o-transform-origin-x: 50%;
    -o-transform-origin-y: 30%;
}
```

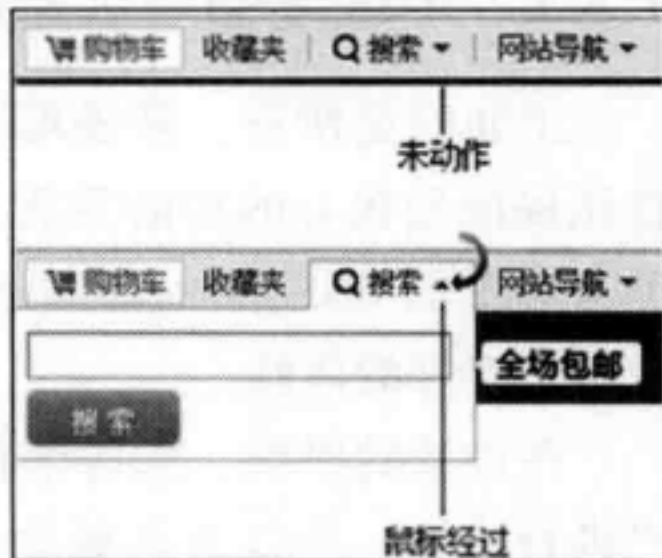


图 1-7 渐进增强制作旋转三角


```

-ms-transform: rotate(180deg);
-ms-transform-origin-x: 50%;
-ms-transform-origin-y: 30%;
transform: rotate(180deg);
transform-origin-x: 50%;
transform-origin-y: 30%;
}

```

采用渐进增强能给现代浏览器用户一个更好的体验,在不支持 CSS3 的 IE 浏览器也能正常使用,只不过体验会大打折扣。

以上只是一个简单的例子,本书后面还会介绍 CSS3 渐进增强的案例,例如 text-shadow 文字阴影、border-radius 圆角属性、box-shadow 盒阴影属性、CSS3 渐变背景和 CSS3 选择器等。

1.4 CSS3 的现状及未来

了解到使用 CSS3 会带来很多好处,哪些网站在使用 CSS3? 这很容易,打开计算机随处可见。既然如此,就一起来看几个网站吧。

1.4.1 谁在使用 CSS3

首先看看新浪微博^①,其中最明显的是圆角的应用,在发表评论的地方还使用了内阴影属性,如图 1-8 所示。

Google 的 UI 也使用了大量 CSS3 特性,看主版面的 Button 效果,这个按钮使用了 CSS3 的阴影、圆角和渐变三种属性,如图 1-9 所示。

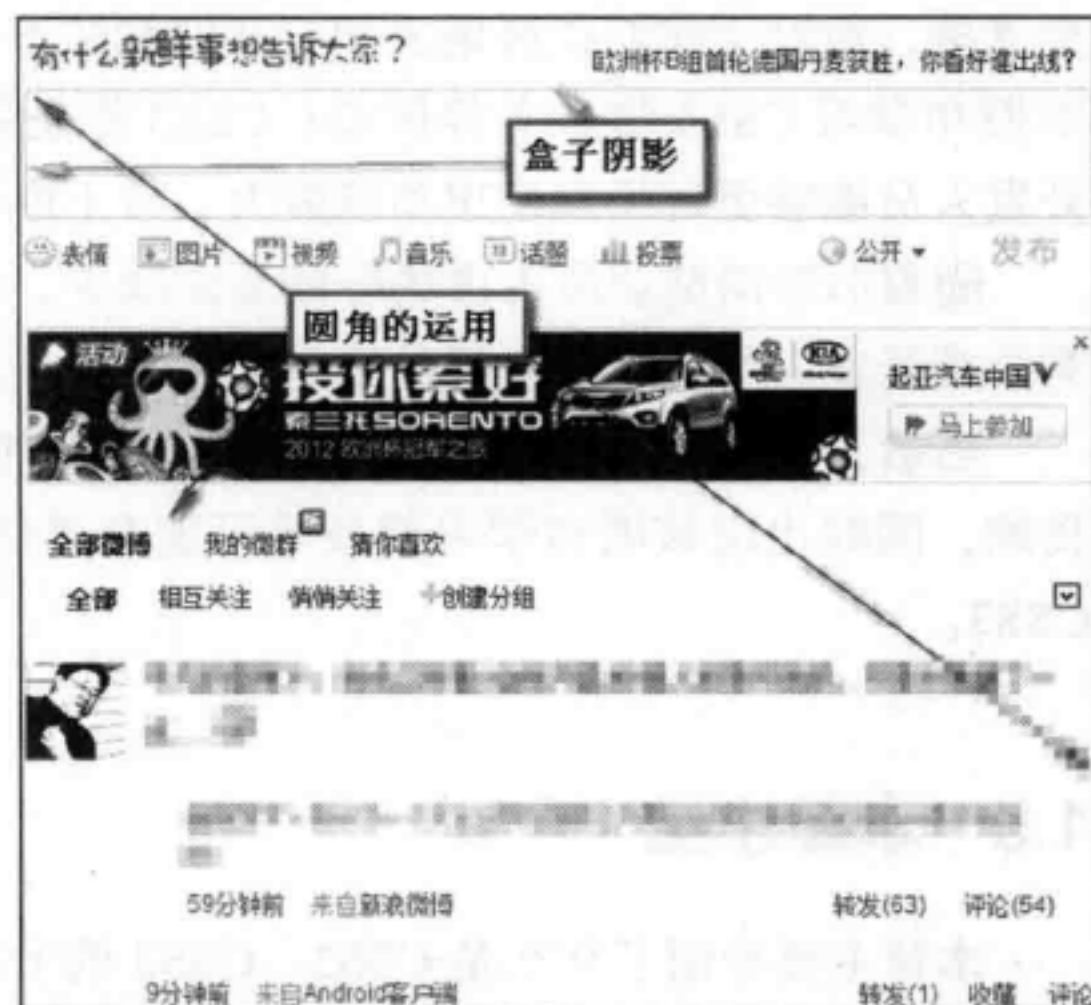


图 1-8 新浪微博



图 1-9 Google 的按钮效果

① 网页地址为 <http://weibo.com/311290005>。

接下来介绍的 Twitter 网站 (<http://twitter.com>) 可以说把 CSS3 运用得出神入化, 登录界面如图 1-10 所示。

整个界面都是使用 CSS3 完成的, 没有使用任何图片、背景渐变、圆角、文本框发光效果等。这些都是 CSS3 特性的一气呵成, 这不得不让我们为 CSS3 强大的特性折服。

国内外使用 CSS3 特性制作网站的案例越来越多, 特别是一些优秀的个人站点, 更是运用得出神入化。

1.4.2 CSS3 的未来

CSS3 无疑对 Web 前端开发带来质的飞跃。虽然目前 CSS3 还没有完全普及, 而且浏览器也不完全支持, 但对于我们积极地去学习和实践并不矛盾, 掌握和学习 CSS3 将是大势所趋。CSS3 将是引导我们进入编写网页精彩世界的先驱技术。开发人员能够更轻松地创建功能强大、易于维护网站。

随着旧版浏览器所占市场份额逐渐减少, 学习 CSS3 技术将更有价值。这是作为一位优秀前端开发人员所必须掌握的技术之一, 也是前端开发人员的大势所趋。

当然, 学习一门新技术不能跟风, 需要理性思考, 这种理性思考并不表示对新技术的畏缩, 同时也应该明白学习新技术可能会遇到的困难和风险。只有这样, 才能更好地驾驭 CSS3。

1.5 本章小结

本章主要介绍了什么是 CSS3、CSS3 的发展状况、新特性, 以及浏览器对 CSS3 的支持状况; 同时, 引进了渐进增强的设计理念。通过本章的学习, 可以对 CSS3 有一个初步的了解。学习 CSS3 的好处有很多, 它能让你始终处于 Web 设计的前沿, 增加你的职业技能和竞争力, 还能帮助你缩短与顶级设计师或开发者的距离。马上开始使用, 然后不断磨砺你的技巧并在职业的道路上不断前进。



图 1-10 Twitter 登录界面

CSS3 选择器

W3C 在 CSS3 的工作草案中把选择器独立出来成为一个模块^①。实际上，选择器是 CSS 知识中的重要部分之一，也是 CSS 的根基。利用 CSS 选择器能不改动 HTML 结构，通过添加不同的 CSS 规则得到不同样式的网页。

2.1 认识 CSS 选择器

要使某个样式应用于特定的 HTML 元素，首先需要找到该元素。在 CSS 中，执行这一任务的表现规则称为 CSS 选择器。它为获取目标元素之后施加样式提供了极大的灵活性。实际上，CSS2.1 已经为大家提供了很多常用的选择器，基本能够满足 Web 设计师常规的设计需求。

2.1.1 CSS3 选择器的优势

既然 CSS 选择器能满足 Web 常规的设计需求，CSS3 选择器有什么优势呢？CSS3 选择器不但支持所有 CSS 选择器，同时新增了独有的选择器，对拥有一定 CSS 基础的开发人员来说，学习 CSS3 选择器是件非常容易的事。

CSS3 选择器在常规选择器的基础上新增了属性选择器、伪类选择器、过滤选择器。可以帮助您在开发中减少对 HTML 类名或 ID 名的依赖，以及对 HTML 元素的结构依赖，使编写代码更加简单轻松。如果学习过 jQuery 选择器，学习 CSS3 选择器会更容易，因为 CSS3 选择器在某些方面和 jQuery 选择器是完全一样的，唯一遗憾的是部分旧版本浏览器

^① 详情参考 <http://www.w3.org/TR/css3-selectors>。

并不支持 CSS3 新增的部分选择器。下面一起来体验 CSS3 选择器。

2.1.2 CSS3 选择器分类

根据所获取页面中元素的不同，把 CSS3 选择器分为五大类：基本选择器、层次选择器、伪类选择器、伪元素和属性选择器。其中，伪类选择器又分为六种：动态伪类选择器、目标伪类选择器、语言伪类、UI 元素状态伪类选择器，结构伪类选择器和否定伪类选择器，如图 2-1 所示。

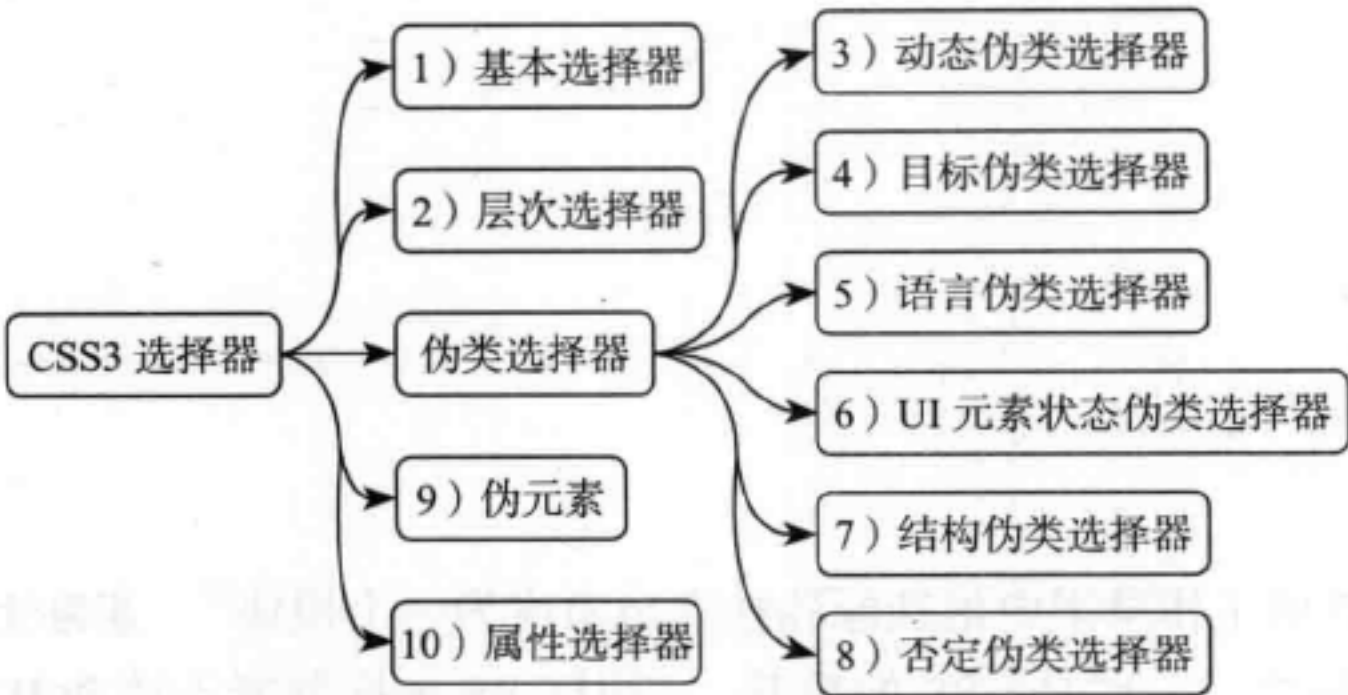


图 2-1 CSS3 选择器分类

下面分别介绍这十种选择器。

2.2 基本选择器

基本选择器是 CSS 中使用最频繁、最基础，也是 CSS 中最早定义的选择器，这部分选择器在 CSS1 中就定义了，为了便于初学者温故而知新，不妨回顾 CSS 的基础选择器。

2.2.1 基本选择器语法

通过基本选择器可以确定 HTML 树形结构中大多数的 DOM 元素节点。其详细说明如表 2-1 所示。






表 2-1 基本选择器语法

选择器	类型	功能描述
*	通配选择器	选择文档中所有的 HTML 元素
E	元素选择器	选择指定的类型的 HTML 元素
#id	ID 选择器	选择指定 ID 属性值为“id”的任意类型元素
.class	类选择器	选择指定 class 属性值为“class”的任意类型的任意多个元素
selector1,selectorN	群组选择器	将每一个选择器匹配的元素集合并

2.2.2 浏览器兼容性

浏览器兼容性如表 2-2 所示。从表中可以看出，浏览器对基本选择器都是一路绿灯通行，大家可以放心使用。

表 2-2 基本选择器的浏览器兼容性

选择器					
*	✓	✓	✓	✓	✓
E	✓	✓	✓	✓	✓
#id	✓	✓	✓	✓	✓
.class	✓	✓	✓	✓	✓
selector1,selectorN	✓	✓	✓	✓	✓

2.2.3 实战体验：使用基本选择器

下面通过示例介绍各种基本选择器在页面中的使用方法。

页面中有一个列表，其中第一个和最后一个设置了 ID 属性，其中部分列表项设置了 class 类名，通过基本选择器来改变元素的样式风格。

新建一个 HTML 文件 2-1.html，加入以下代码。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>使用 CSS3 基本选择器</title>
  <style type="text/css">
    *{margin: 0;padding:0;}
    .clearfix:after,.clearfix:before{display:table;content:""}
    .clearfix:after{clear:both;overflow:hidden}
    .demo { width: 250px; border: 1px solid #ccc; padding: 10px;margin: 20px auto}
    li {list-style:none outside none; float: left; height: 20px;
      line-height: 20px; width: 20px;border-radius: 10px;
      text-align: center; background: #f36; color: green; margin-right: 5px; }
  </style>
</head>
<body>
  <ul class="clearfix demo">
    <li class="first" id="first">1</li>
    <li class="active">2</li>
    <li class="important item">3</li>
    <li class="important">4</li>
    <li class="item">5</li>
    <li>6</li>
    <li>7</li>
    <li>8</li>
    <li>9</li>
    <li class="last" id="last">10</li>
  </ul>
</body>
</html>
```

上面代码使用了基本选择器，首先看看页面的初步效果（背景为粉红色），如图 2-2 所示。

下面通过图解的方法说明 CSS3 基本选择器的使用方法。



☆图 2-2 页面初步效果[⊖]

2.2.4 通配选择器

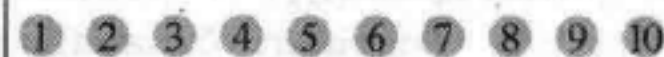
通配选择器 (*) 用来选择所有元素，当然也可以选择某个元素下的所有元素。如：

```
* {margin: 0;padding:0;}
```

上面一行代码大家在 Reset 样式文件中经常看到，表示所有元素的 margin 和 padding 都设置为 0。为了更好地说明问题，通过 CSS3 选择器中的通配选择器来改变列表中所有子项的背景色设置为 orange。

```
*{margin: 0;padding:0;}
.clearfix:after,.clearfix:before{display:table;content:""}
.clearfix:after{clear:both;overflow:hidden}
.demo { width: 250px; border: 1px solid #ccc; padding: 10px;margin: 20px auto}
li {list-style:none outside none; float: left; height: 20px;
    line-height: 20px; width: 20px;border-radius: 10px; text-align: center;
background: #f36; color: green; margin-right: 5px; }
.demo * {background: orange}
```

此时元素类名为 demo 下的所有元素都将背景色设置为橙色，如图 2-3 所示。



☆图 2-3 通配选择器使用效果

2.2.5 元素选择器

元素选择器 (E) 是 CSS 选择器中最常见、最基本的选择器。文档的元素包括 html、body、p、iv 等，如示例中 ul、li 也属于 HTML 元素。接下来通过 ul 元素选择器改变整个列表的背景色。

```
*{margin: 0;padding:0;}
.clearfix:after,.clearfix:before{display:table;content:""}
.clearfix:after{clear:both;overflow:hidden}
.demo { width: 250px; border: 1px solid #ccc; padding: 10px;margin: 20px auto}
li {list-style:none outside none; float: left; height: 20px; line-height: 20px;
    width: 20px;border-radius: 10px; text-align: center; background: #f36;
    color: green; margin-right: 5px; }
.demo *{background: orange}
ul {background:grey}
```

此时列表 ul 的背景色将变成灰色，如图 2-4 所示。



☆图 2-4 元素选择器使用效果

2.2.6 ID 选择器

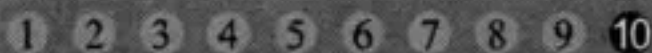
在使用 ID 选择器 (#id) 之前，需要在 HTML 文档中给对应的元素设置 id 属性并设置

[⊖] 有☆号的图示附有彩色图，后同。

其值,才能找到对应的元素。ID 选择器具有唯一性,在一个页面中不会同时出现 id 相同的属性值。在 CSS 样式中使用 id 选择器时,需要在 id 属性值的前面加上“#”号,如 (#id)。在下面这个示例中,可以轻松地看到列表中的第一项和最后一项分别定义了一个 id,其值分别为“first”和“last”。使用这两个 id 值来改变列表项中第一个和最后一个列表项的背景色和前景色,代码如下。

```
*{margin: 0;padding:0;}
.clearfix:after,.clearfix:before{display:table;content:""}
.clearfix:after{clear:both;overflow:hidden}
.demo { width: 250px; border: 1px solid #ccc; padding: 10px;margin: 20px auto}
li {list-style:none outside none; float: left; height: 20px; line-height: 20px;
width: 20px;border-radius: 10px; text-align: center; background: #f36;
color: green; margin-right: 5px; }
.demo *{background: orange}
ul {background:grey}
#first{background:lime;color:#000}
#last {background:#000;color:lime}
```

页面效果如图 2-5 所示。



2.2.7 类选择器

☆图 2-5 ID 选择器使用效果

类选择器 (.class) 是以独立于文档元素的方式来指定元素样式。使用方法与 ID 选择器极其相似,首先在 HTML 给需要的元素定义 class 属性,并为其设置属性值。其与 ID 选择器有一个很大不同之处。“类选择器在一个页面中可以有多个相同的类名,而 ID 选择器其 ID 值在整个页面中是唯一的一个”。同样,看看如何通过类选择器来改变元素的样式。

```
*{margin: 0;padding:0;}
.clearfix:after,.clearfix:before{display:table;content:""}
.clearfix:after{clear:both;overflow:hidden}
.demo { width: 250px; border: 1px solid #ccc; padding: 10px;margin: 20px auto}
li {list-style:none outside none; float: left; height: 20px;
line-height: 20px; width: 20px;border-radius: 10px;
text-align: center; background: #f36; color: green;
margin-right: 5px; }
.demo *{background: orange}
ul {background:grey}
#first{background:lime;color:#000}
#last {background:#000;color:lime}
.item {background: green;color: #fff;font-weight:bold}
```

页面效果就变成图 2-6 所示的风格了。

上面是类选择器的简单使用,其实类选择器还有一种使用方法,就是多类选择器。通过两个或两个以上类选择器合并,来定义有别于一个类名的元素效果。



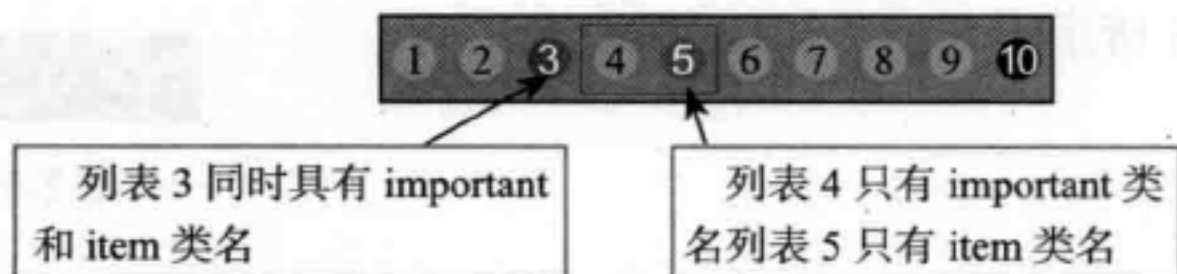
☆图 2-6 类选择器使用效果

```

*{margin: 0;padding:0;}
.clearfix:after,.clearfix:before{display:table;content:""}
.clearfix:after{clear:both;overflow:hidden}
.demo { width: 250px; border: 1px solid #ccc; padding: 10px;margin: 20px auto}
li {list-style:none outside none; float: left; height: 20px;
    line-height: 20px; width: 20px;border-radius: 10px;
    text-align: center; background: #f36; color: green;
    margin-right: 5px; }
.demo *{background: orange}
ul {background:grey}
#first{background:lime;color:#000}
#last {background:#000;color:lime}
.item {background: green;color: #fff;font-weight:bold}
.item.important {background:red;}

```

使用多类选择器时，大家需要注意，如果一个多类选择器包含的类名中其中有一个不存在，这个选择器将无法找到相匹配的元素，正如上面的代码，其只能匹配 li 元素同时具有“item”和“important”的元素，而只有其中任何一个类名都将无法匹配，如图 2-7 所示。



☆图 2-7 多类名选择器使用效果



注意

IE 6 选择器并不支持多类名选择器，其将以末尾类名为准，大家使用时千万小心。


由于类名在一个 HTML 文档中可以同时存在于不同的元素标签上。换句话说，在一个 HTML 文档中，div 可以有类名“block”，ul 也可以有类名“block”，但有时在 Web 的页面开发中，仅需要对 ul 为“block”定义样式，此时仅采用类名选择器并不能达到需要的效果，其实 CSS 选择器还支持带有标签的类名选择器“ul.block”。

```
ul.block{background:#ccc;}
```

上面的代码只匹配 class 属性包含“block”的所有 ul 元素，但其他任何类型的元素都不匹配，包括有“block”类名的元素。简而言之，“ul.block”只会匹配 ul 元素，并且有一个类名“block”。不符合这两个条件的任何一个都不能与选择器匹配。

2.2.8 群组选择器

群组选择器 (selector1,selectorN) 是将具有相同样式的元素分组在一起，每个选择器之间用逗号 (,) 隔开，例如“selector1,selector2,...,selectorN”。这个逗号告诉浏览器，规则中包含多个不同的选择器，省去逗号就成了后代选择器，这一点大家在使用中千万要小心。

 **注意** “selector1,selectorN”是群组选择器，表示选择匹配为 selector1 和 selectorN 的所有元素；“selector1 selectorN”是后代选择器（后面介绍），表示选择器 selectorN 所有元素为 selector1 的后代元素。

2.3 层次选择器

层次选择器通过 HTML 的 DOM 元素间的层次关系获取元素，其主要的层次关系包括后代、父子、相邻兄弟和通用兄弟几种关系，通过其中某类关系可以方便快捷地选定需要的元素。

2.3.1 层次选择器语法

层次选择器是一个非常好的选择器，也是大家常用的选择器，其详细说明如表 2-3 所示。






表 2-3 层次选择器语法

选择器	类型	功能描述
E F	后代选择器（包含选择器）	选择匹配的 F 元素，且匹配的 F 元素被包含在匹配的 E 元素内
E > F	子选择器	选择匹配的 F 元素，且匹配的 F 元素是所匹配的 E 元素的子元素
E + F	相邻兄弟选择器	选择匹配的 F 元素，且匹配的 F 元素紧位于匹配的 E 元素后面
E ~ F	通用选择器	选择匹配的 F 元素，且位于匹配的 E 元素后的所有匹配的 F 元素

2.3.2 浏览器兼容性

浏览器的兼容性如表 2-4 所示。从表中可以看出，子选择器、相邻兄弟选择器和通用兄弟选择器要 IE 7 以及其以上版本才支持，随着 IE 6 的慢慢消失，层次选择可以放心使用。

表 2-4 层次选择器的浏览器兼容性

选择器					
E F	√	√	√	√	√
E > F	7+√	√	√	√	√
E + F	7+√	√	√	√	√
E ~ F	7+√	√	√	√	√

2.3.3 实战体验：使用层次选择器选择元素

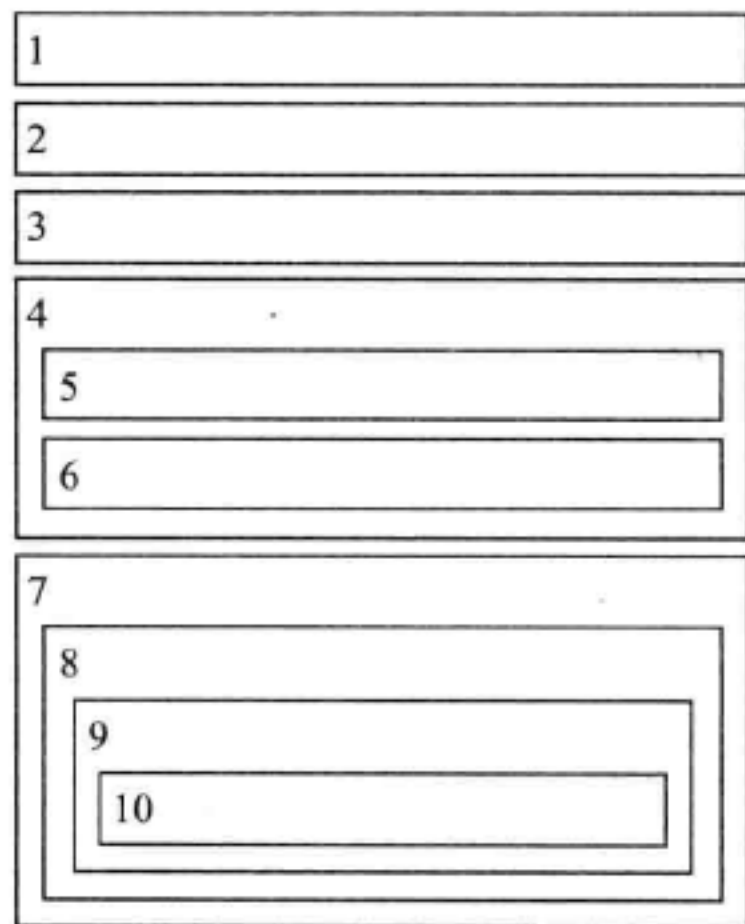
在层次选择器中，后代选择器与子选择器是比较常用的，而对于相邻兄弟选择器和通用兄弟选择器而言，平时大家并不常使用，特别是 CSS3 选择器中新增的通用兄弟选择器。下面通过示例来演示各种层次选择器在页面中如何选择 HTML 的 DOM 元素的方法。

页面中有 10 个 div 元素，其中第四个 div 中包含了 2 个 div，另外第七个 div 包含了第八个，而第八个 div 又包含了第九个 div，并且第九个 div 包含了第十个 div。下面通过层次

选择器来改变 div 的样式风格。

新建一个 HTML 文件 2-2.html，代码如下。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title> 使用 CSS3 层次选择器 </title>
  <style type="text/css">
    *{margin: 0;padding:0;}
    body {width: 300px;margin: 0 auto;}
    div{margin:5px;padding:5px;border: 1px solid #ccc;}
  </style>
</head>
<body>
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4
    <div>5</div>
    <div>6</div>
  </div>
  <div>7
    <div>8
      <div>9
        <div>10</div>
      </div>
    </div>
  </div>
</body>
</html>
```



在具体使用层选择器之前，先来看看页面的初步效果，如图 2-8 所示。

图 2-8 页面初步效果

为了更好地理清这些 div 的层次关系，可以先将示例中的 body 部分画一个 DOM 树形草图，如图 2-9 所示。

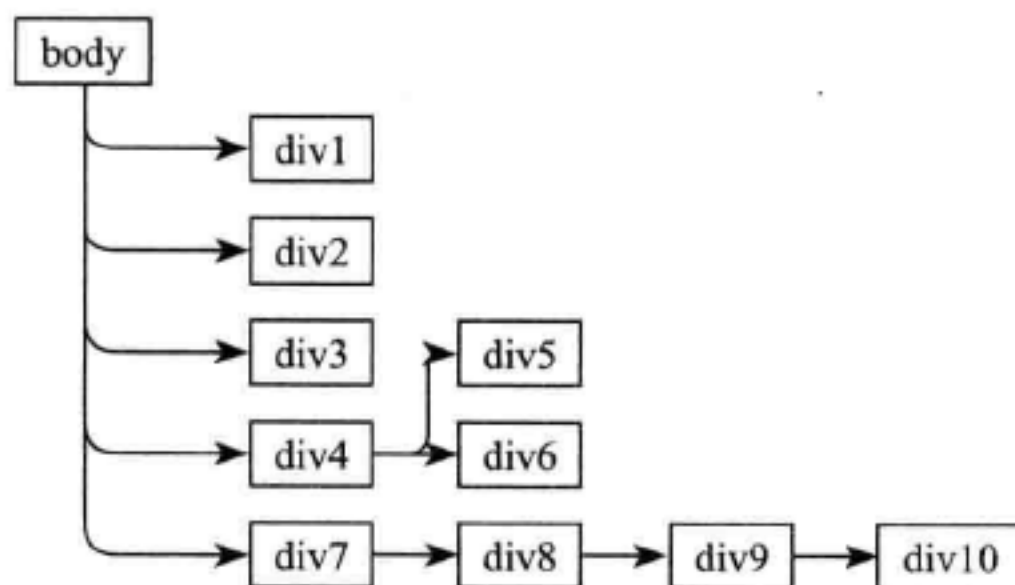


图 2-9 body 的树形结构

2.3.4 后代选择器

后代选择器 (E F) 也称为包含选择器, 作用就是可以选择某元素的后代元素。例如 “E F”, E 为祖先元素, F 为后代元素, 表达的意思就是选择 E 元素的所有后代 F 元素。这里的 F 元素不管是 E 元素的子元素、孙辈元素或者更深层次的关系, 都将被选中。换句话说, 不论 F 在 E 中有多少层级关系, F 元素都将被选中。接下来使用后代选择器来改变其背景色。

```
*{margin: 0;padding:0;}
body {width: 300px;margin: 0 auto;}
div{margin:5px;padding:5px;border: 1px solid #ccc;}
div div {background: orange;}
```

一起来看使用后代选择器改变了哪几个 div 的背景色, 如图 2-10 所示。

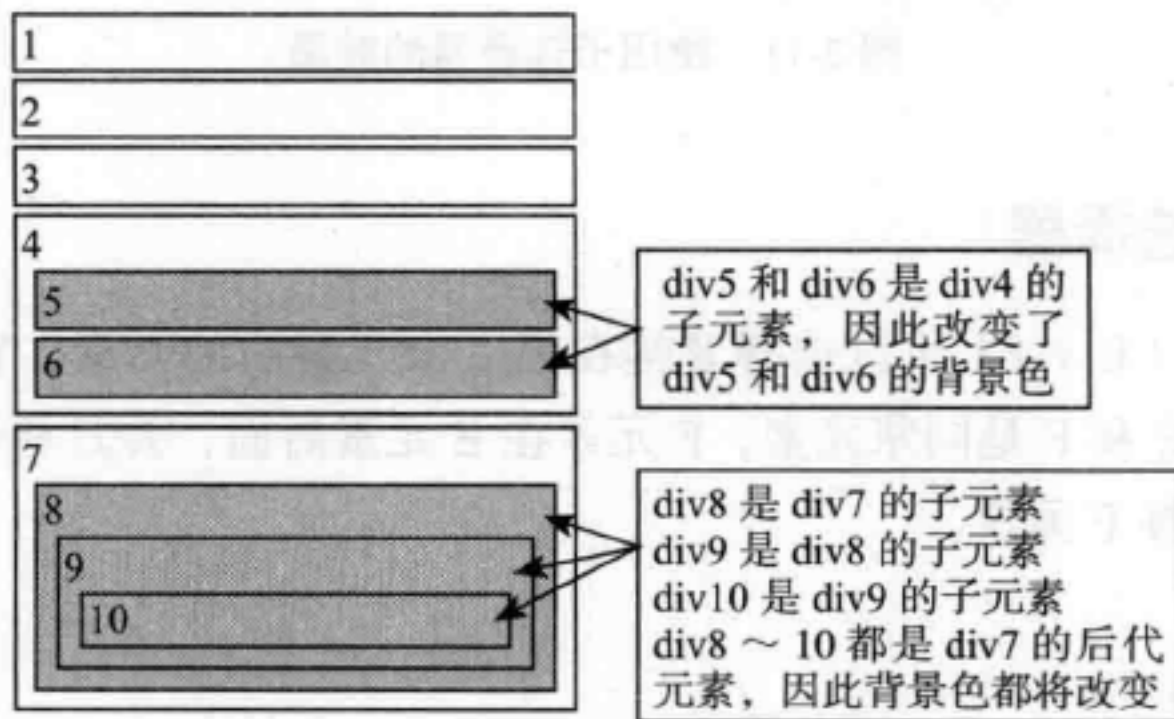


图 2-10 使用后代选择器的效果



注意

后代选择器两选择符之间必须以空格隔开, 中间不能有任何其他符号插入。

2.3.5 子选择器

子选择器 (E > F) 只能选择某元素的子元素, 其中 E 为父元素, 而 F 为子元素, 其中 E>F 表示选择了 E 元素下所有子元素 F。这与后代选择器 (E F) 不一样, 在后代选择器中 F 是 E 的后代元素, 而在 E>F 中 F 仅仅是 E 的子元素而已。接下来通过例子, 选择器改变 body 下的子元素 div 的背景色。

```
*{margin: 0;padding:0;}
body {width: 300px;margin: 0 auto;}
div{margin:5px;padding:5px;border: 1px solid #ccc;}
div div {background: orange;}
body > div {background: green;}
```

通过上面子选择器的运用, 你能猜出 body 的 10 个 div 中, 哪几个背景色将变成绿色

呢？来看图 2-11 的效果。

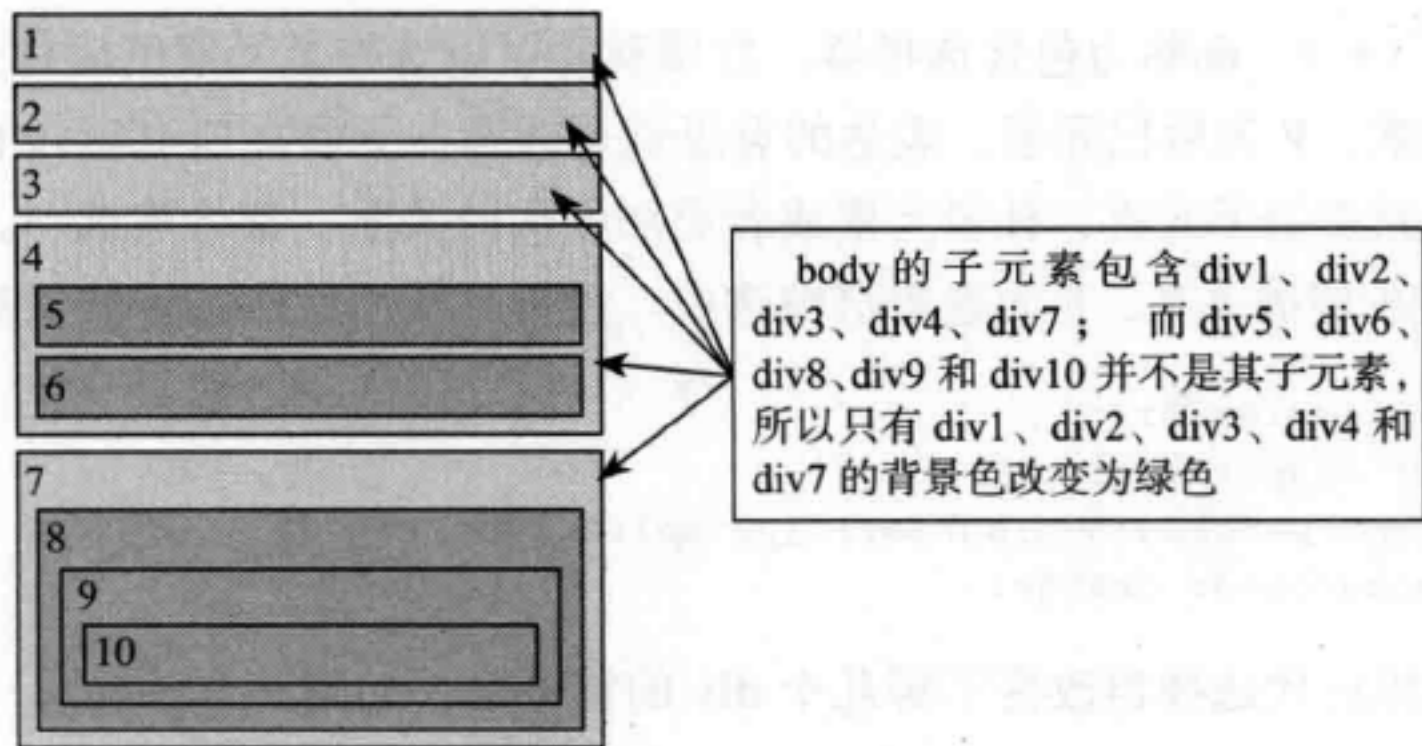


图 2-11 使用子选择器的效果

2.3.6 相邻兄弟选择器

相邻兄弟选择器 (E + F) 可以选择紧接在另一个元素后的元素，它们具有一个相同的父元素。换句话说，E 和 F 是同辈元素，F 元素在 E 元素后面，并且相邻，这样就可以使用相邻兄弟选择器来选择 F 元素。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>使用 CSS3 层次选择器</title>
  <style type="text/css">
    *{margin: 0;padding:0;}
    body {width: 300px;margin: 0 auto;}
    div{margin:5px;padding:5px;border: 1px solid #ccc;}
    div div {background: orange;}
    body > div {background: green;}
    .active + div {background: lime;}
  </style>
</head>
<body>
  <div class="active">1</div>
  <!-- 为了说明相邻兄弟选择器，在此处添加一个类名 active -->
  <div>2</div>
  <div>3</div>
  <div>4
    <div>5</div>
    <div>6</div>
  </div>
  <div>7
    <div>8
```



```

    <div>9
      <div>10</div>
    </div>
  </div>
</body>
</html>

```

此时第二个 div 的背景色将变成“lime”色，如图 2-12 所示。

2.3.7 通用兄弟选择器

通用兄弟选择器（E ~ F）是 CSS3 新增的，用于选择某元素后面的所有兄弟元素，它们和相邻兄弟选择器类似，需要在同一个父元素之中。也就是说，E 和 F 元素都是同辈元素，并且 F 元素在 E 元素之后，E ~ F 将选中 E 元素后面的所有 F 元素。如下面的代码所示。

```

*{margin: 0;padding:0;}
body {width: 300px;margin: 0 auto;}
div{margin:5px;padding:5px;border:
1px solid #ccc;}
div div {background: orange;}
body > div {background: green;}
.active + div {background: lime;}
.active ~ div {background: red;}

```

这样，只要是 div.active 的兄弟元素 div，并且在 div.active 之后，其背景色将变成红色，如图 2-13 所示。

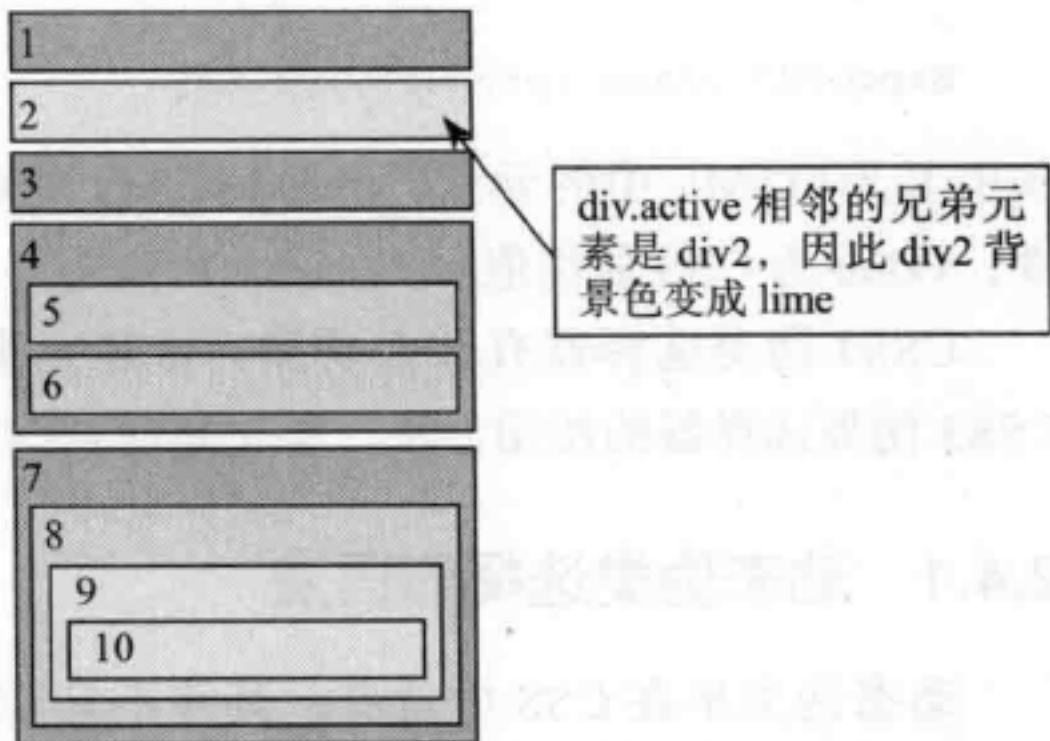
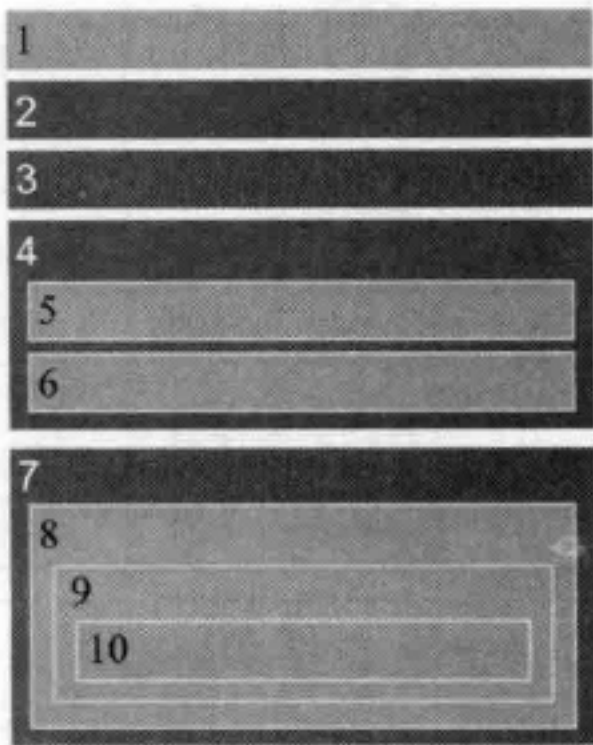


图 2-12 使用相邻兄弟选择器效果



☆图 2-13 使用通用兄弟选择器效果



注意

通用兄弟选择器选中的是与 E 元素相邻的后面兄弟元素 F，其选中的是一个或多个元素；而相邻兄弟选择器选中的仅是与 E 元素相邻并且紧挨的兄弟元素 F，其选中的仅是一个元素。

2.4 动态伪类选择器

伪类选择器对于大家来说最熟悉的莫过于“:link”、“:visited”、“:hover”、“:active”，因为这些是大家平时常用到的伪类选择器。而 CSS3 的伪类选择器可以分成六种：动态伪类选择器、目标伪类选择器、语言伪类选择器、UI 状态伪类选择器、结构伪类选择器和否定

伪类选择器。

伪类选择器语法书写时和其他的 CSS 选择器写法有所区别，都以冒号 (:) 开头。例如：

```
E:pseudo-class {property:value}
```

其中 E 为 HTML 中的元素；pseudo-class 是 CSS 的伪类选择器名称；property 是 CSS 的属性；value 为 CSS 属性值。


CSS3 伪类选择器有什么功能？选定元素能带来什么便利？带着这些问题，依次学习 CSS3 伪类选择器的使用方法，首先是动态伪类选择器。

2.4.1 动态伪类选择器语法

动态伪类早在 CSS 中就有，其并不是 CSS3 才有的，动态伪类并不存在于 HTML 中，只有当用户和网站交互的时候才能体现出来。动态伪类包含两种，第一种是在链接中常看到的锚点伪类，另一种为用户行为伪类。其详细说明如表 2-5 所示。

表 2-5 动态伪类选择器语法






选择器	类型	功能描述
E:link	链接伪类选择器	选择匹配的 E 元素，而且匹配元素被定义了超链接并未被访问过。常用于链接锚点上
E:visited	链接伪类选择器	选择匹配的 E 元素，而且匹配元素被定义了超链接并已被访问过。常用于链接锚点上
E:active	用户行为伪类选择器	选择匹配的 E 元素，且匹配元素被激活。常用于锚点与按钮上
E:hover	用户行为伪类选择器	选择匹配的 E 元素，且用户鼠标在停留在元素 E 上。IE 6 及以下浏览器仅支持 a:hover
E:focus	用户行为伪类选择器	选择匹配的 E 元素，且匹配的元素获得焦点

 **注意** 锚点伪类的设置必须遵守一个“爱恨原则” LoVe/HaTe，也就是“link-visited-hover-active”。另外，在 IE 6、IE 7(Q)、IE 8(Q) 中，a:hover、a:active 和 a:visited 并没有按照规范描述的算法来计算它们的针对性，而是根据链接的实际状态来决定使用哪个规则集里的声明。它们三个的针对性比 a:link 强，详细参阅 <http://www.w3help.org/zh-cn/causes/RS3005>。






2.4.2 浏览器兼容性

浏览器兼容性如表 2-6 所示。

表 2-6 动态伪类选择器浏览器兼容性

选择器					
E:link	√	√	√	√	√
E:visited	√	√	√	√	√

(续)

选择器					
E:active	8+√	√	√	√	√
E:hover	√	√	√	√	√
E:focus	8+√	√	√	√	√



注意 E:hover 在 IE 6 浏览器中仅支持链接锚点 a:hover。

2.4.3 实战体验：美化按钮

在众多网站上按钮在不同状态下效果不一，用以增强用户体验，这也是一种非常好的设计体验与细节。实现并不复杂，下面一起来看 Twitter 的 Bootstrap^① 如何美化按钮。

实现页面中按钮在不同行为状态下的样式风格，常见的行为状态如默认状态、悬浮状态、用户点击时状态和按钮获得焦点状态。

通过 Twitter 的 Bootstrap 制作的按钮效果如图 2-14 所示。

根据用户的行为不同，按钮效果可以分为：默认状态、悬浮状态、点击时状态、焦点状态和点击后状态，可以按照 CSS3 的动态伪选择器，在不同状态下给按钮赋予不同的样式风格。

根据这一设计思路，可以轻松美化按钮，看下面的示例代码，演示的效果如图 2-14 所示。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title> 使用动态伪类选择器美化按钮 </title>
  <style type="text/css">
    .download-info {
      text-align: center;
    }
    /* 默认状态下的按钮效果 */
    .btn {
      background-color: #0074cc;
      *background-color: #0055cc;
      /*CSS3 渐变制作背景图片*/
      background-image: -ms-linear-gradient(top, #0088cc, #0055cc);
      background-image: -webkit-gradient(linear, 0 0, 0 100%,
        from(#0088cc), to(#0055cc));
      background-image: -webkit-linear-gradient(top, #0088cc, #0055cc);
      background-image: -o-linear-gradient(top, #0088cc, #0055cc);
```

默认状态

View project on GitHub

悬浮状态

View project on GitHub

点击状态

View project on GitHub

☆图 2-14 美化按钮效果

① 参考 <http://twitter.github.com/bootstrap/>。

```

background-image: -moz-linear-gradient(top, #0088cc, #0055cc);
background-image: linear-gradient(top, #0088cc, #0055cc);
background-repeat: repeat-x;
display: inline-block;
*display: inline;
border: 1px solid #cccccc;
*border: 0;
border-color: #ccc;
/*CSS3 的色彩模块*/
border-color: rgba(0, 0, 0, 0.1) rgba(0, 0, 0, 0.1) rgba(0, 0, 0, 0.25);
border-radius: 6px;
color: #ffffff;
cursor: pointer;
font-size: 20px;
font-weight: normal;
filter: progid:dximagetransform.microsoft.gradient
        (startColorstr='#0088cc', endColorstr='#0055cc', GradientType=0);
filter: progid:dximagetransform.microsoft.gradient(enabled=false);
line-height: normal;
padding: 14px 24px;
text-align: center;
/*CSS3 文字阴影特性*/
text-shadow: 0 -1px 0 rgba(0, 0, 0, 0.25);
text-decoration: none;
vertical-align: middle;
*zoom: 1;
}
/* 悬浮状态下按钮效果 */
.btn:hover {
background-position: 0 -15px;
background-color: #0055cc;
*background-color: #004ab3;
color: #ffffff;
text-decoration: none;
text-shadow: 0 -1px 0 rgba(0, 0, 0, 0.25);
/*CSS3 动画效果*/
-webkit-transition: background-position 0.1s linear;
-moz-transition: background-position 0.1s linear;
-ms-transition: background-position 0.1s linear;
-o-transition: background-position 0.1s linear;
transition: background-position 0.1s linear;
}
/* 点击时按钮效果 */
.btn:active {
background-color: #0055cc;
*background-color: #004ab3;
background-color: #004099 \9;
background-image: none;
outline: 0;
/*CSS3 盒子阴影特性*/
box-shadow: inset 0 2px 4px rgba(0, 0, 0, 0.15),

```



```

        0 1px 2px rgba(0, 0, 0, 0.05);
    color: rgba(255, 255, 255, 0.75);
}
/* 获得焦点按钮效果 */
.btn:focus {
    outline: thin dotted #333;
    outline: 5px auto -webkit-focus-ring-color;
    outline-offset: -2px;
}
</style>
</head>
<body>
    <div class="download-info">
        <a href="#" class="btn">View project on GitHub</a>
    </div>
</body>
</html>

```

这个美化按钮实例涉及一些 CSS3 的特性，此时不懂不要害怕，本书后续章节中会为大家逐一介绍。在此例中只需要知道哪些属性是 CSS3 的特性就足够了。同时实例中采用了“渐进增强，优雅降级”，在不支持 CSS3 的浏览器中，同样可以看到按钮在不同状态下的效果，只是失去了渐变、阴影等效果，但并不影响网站的功能与用户的体验，IE 8 下的效果如图 2-15 所示。



☆图 2-15 IE 8 下的按钮效果

2.5 目标伪类选择器

目标伪类选择器“:target”是众多实用的 CSS3 特性中的一个，用来匹配文档（页面）的 URI^①中某个标志符的目标元素。具体来说，URI 中的标志符通常会包含一个井号（#），后面带有一个标志符名称，例如“#contact”“:target”就是用来匹配 ID 为“contact”的元素的。换种说法，在 Web 页面中，一些 URL 拥有片段标识符，它由一个井号（#）后跟一个锚点或元素 ID 组合而成，可以链接到页面的某个特定元素。“:target”伪类选择器选取链接的目标元素，然后供定义样式。

2.5.1 目标伪类选择器语法

目标伪类选择器的语法如表 2-7 所示。

表 2-7 目标伪类选择器语法

选择器	功能描述
E:target	选择匹配 E 的所有元素，且匹配元素被相关 URL 指向








注意 目标伪类选择器是动态选择器，只有存在 URL 指向该匹配元素时，样式效果才会生效。

① 参考 http://en.wikipedia.org/wiki/Uniform_Resource_Identifier#cite_note-1。

2.5.2 浏览器兼容性

浏览器兼容性如表 2-8 所示。

表 2-8 目标伪类选择器浏览器兼容性

选择器					
E:target	9 + √	√	√	9.6 + √	√

从表 2-8 可知，目标伪类选择器在 IE 8 及之前版本不被支持，但 IE 用户点击目录里的链接仍将跳转到相应的标题，只是标题不会高亮显示。



在包含更多内容的页面中，高亮显示效果的确能给用户带来极好的体验。如果页面效果需要兼容 IE 低版本浏览器，就要用到 JavaScript。这里有一些资源供大家参考。

- “Suckerfish :target”，作者：Patrick Griffiths 和 Dan Webb，网址如下。
<http://www.htmldog.com/articles/suckerfish/target>
- “Improving the usability of within-page links”，作者：Bruce Lawson，网址如下。
<http://dev.opera.com/articles/view/improving-the-usability-of-within-page-l/>

2.5.3 实战体验：制作手风琴效果

以前制作手风琴效果（Accordion）需要依赖 JavaScript 脚本。CSS3 的目标伪选择器可不使用任何 JavaScript 代码实现手风琴效果。

页面中有三个区块，默认状态只显示三个区块的标题，点击其中一个标题时，其对应的内容就会显示；点击另一个标题时，对应区块内容将显示，而前一块内容将隐藏。目标伪类选择器制作的页面效果如图 2-16 所示。

通过目标伪类选择器 “E:target”，显示和隐藏不同栏目的内容，从而实现手风琴效果。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>垂直手风琴</title>
  <style type="text/css">
    .accordionMenu {
      background: #fff;
      color: #424242;
      font: 12px Arial, Verdana, sans-serif;
      margin: 0 auto;
      padding: 10px;
      width: 500px;
    }
    .accordionMenu h2 {
      margin: 5px 0;
    }
  </style>
</head>
<body>
  <div>
    <h2>Brand</h2>
    <div>content for Brand</div>
  </div>
  <div>
    <h2>Promotion</h2>
    <div>content for Promotion</div>
  </div>
  <div>
    <h2>Event</h2>
    <div>content for Event</div>
  </div>
  </body>
</html>
```

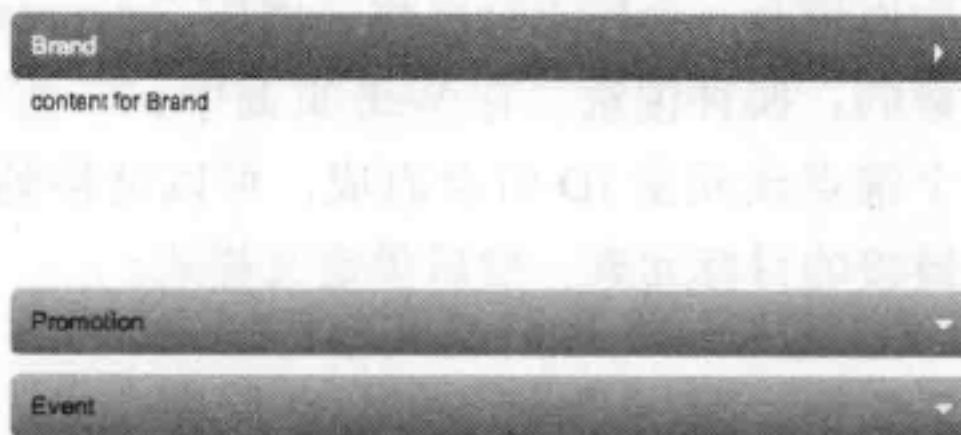


图 2-16 手风琴效果


```

padding:0;
position: relative;
}
.accordionMenu h2:before { /* 制作向下三角效果 */
border: 5px solid #fff;
border-color: #fff transparent transparent;
content:"";
height: 0;
position:absolute;
right: 10px;
top: 15px;
width: 0;
}
.accordionMenu h2 a { /* 制作手风琴标题栏效果 */
background: #8f8f8f;
background: -moz-linear-gradient( top, #cecece, #8f8f8f);
background: -webkit-gradient(linear, left top, left bottom,
    from(#cecece), to(#8f8f8f));
background: -webkit-linear-gradient( top, #cecece, #8f8f8f);
background: -o-linear-gradient( top, #cecece, #8f8f8f);
background: linear-gradient( top, #cecece, #8f8f8f);
border-radius: 5px;
color:#424242;
display: block;
font-size: 13px;
font-weight: normal;
margin:0;
padding:10px 10px;
text-shadow: 2px 2px 2px #aeaeae;
text-decoration:none;
}
.accordionMenu :target h2 a, /* 目标标题的效果 */
.accordionMenu h2 a:focus,
.accordionMenu h2 a:hover,
.accordionMenu h2 a:active {
background: #2288dd;
background: -moz-linear-gradient( top, #6bb2ff, #2288dd);
background: -webkit-gradient(linear, left top, left bottom,
    from(#6bb2ff), to(#2288dd));
background: -webkit-linear-gradient( top, #6bb2ff, #2288dd);
background: -o-linear-gradient( top, #6bb2ff, #2288dd);
background: linear-gradient( top, #6bb2ff, #2288dd);
color:#FFF;
}
.accordionMenu p { /* 标题栏对应的内容 */
margin:0;
height: 0; /* 默认栏目内容高度为 0, 达到隐藏效果 */
overflow: hidden;
padding:0 10px;
-moz-transition: height 0.5s ease-in;
-webkit-transition: height 0.5s ease-in;
}

```

```

-o-transition: height 0.5s ease-in;
transition: height 0.5s ease-in;
}
/* 这部分是显示内容的关键代码 */
.accordionMenu :target p { /* 展开对应目标内容 */
    height:100px; /* 显示对应目标栏内容 */
    overflow: auto;
}
.accordionMenu :target h2:before { /* 展开时标题三角效果 */
    border-color: transparent transparent transparent #fff;
}
</style>
</head>
<body>
    <div class="accordionMenu">
        <div class="menuSection" id="brand">
            <h2><a href="#brand">Brand</a></h2>
            <p>Lorem ipsum dolor...</p>
        </div>
        <div class="menuSection" id="promotion">
            <h2><a href="#promotion">Promotion</a></h2>
            <p>Lorem ipsum dolor sit amet...</p>
        </div>
        <div class="menuSection" id="event">
            <h2><a href="#event">Event</a></h2>
            <p>Lorem ipsum dolor sit amet...</p>
        </div>
    </div>
</body>
</html>

```

维基百科的官网^①上就运用了目标伪类选择器来高亮显示脚注，如图 2-17 所示。



☆图 2-17 目标伪类选择器高亮显示区块的运用

① 网址为 http://zh.wikipedia.org/wiki/统一资源标志符#cite_note-0。


点击注解的上标链接时，其对应的注解内容区块就会高亮显示，以便用户在众多内容中找到对应的注解内容，方便用户阅读，而实现此功能仅一行代码就完成了。

```
ol.references > li:target, sup.reference:target, cite:target {
    background-color: #def;
}
```

除了能制作手风琴效果、高亮显示脚注之外，目标伪类选择器还可以用在以下场景，如表 2-9 所示。

表 2-9 “:target” 应用场景

效果	地址
高亮显示区块	http://www.red-team-design.com/get-to-know-your-css3-target-pseudo-class
从相互层叠的盒容器或图片中突出显示其中一项	http://virtuelvis.com/gallery/css3/target/interface.html
tabs 效果	http://css-tricks.com/css3-tabs/
幻灯片效果	http://designmodo.com/slider-css3/
灯箱效果	http://www.decodeize.com/demos/CSS3-target-pseudo-class/gallery.html
相册效果	http://www.ie7nomore.com/fun/scroll/

 **注意** 其中几项效果使用 JavaScript 制作效果会更好，因为纯 CSS 的版本可能存在潜在的易用性和可用性问题。

2.6 语言伪类选择器

使用语言伪类选择器来匹配使用语言的元素是非常有用的，特别是用于多语言版本的网站，其作用更是明显。可以使用他来根据不同语言版本设置页面的字体风格。

2.6.1 语言伪类选择器语法

语言伪类选择器是根据元素的语言编码匹配元素。这种语言信息必须包含在文档中，或者与文档关联，不能从 CSS 指定。为文档指定语言，有两种方法可以表示。如果使用 HTML 5，直接可以设置文档的语言。例如：

```
<!DOCTYPE HTML>
<html lang="en-US">
```

另一种方法就是手工在文档中指定 lang 属性，并设置对应的语言值。例如：

```
<body lang="fr">
```

语言伪类选择器允许为不同的语言定义特殊的规则，这在多语言版本的网站用起来是






特别的方便。

`E:lang(language)` 表示选择匹配 `E` 的所有元素，且匹配元素指定了 `lang` 属性，而且其值为 `language`。

表 2-10 语言伪类选择器的浏览器兼容性

2.6.2 浏览器兼容性

浏览器兼容性如表 2-10 所示。

选择器					
<code>E:target</code>	8 + √	√	√	9.2 + √	√

语言伪类选择器在 IE 7 及以下版本中还不被支持，对于追求完美的设计师来说，又有点畏惧，不敢使用。其实也不是没有办法，可以为不同的浏览器（IE 6 和 IE 7）采用不同的方法来实现。

- 对于 IE 6 浏览器，给引文元素在不同版本的时候设置不同的类名，例在英文版本下设置类名 `.en`，而在法文版本下设置类名为 `.fr`，就可以通过类名给引文定义不同的样式。
- 对于 IE 7 浏览器，也可以采用 IE 6 浏览器的方法。如果不考虑 IE 6 浏览器，可以使用属性选择器中的 `E[foo="en"]` 选择器为不同语言版本的引文设置不同样式。

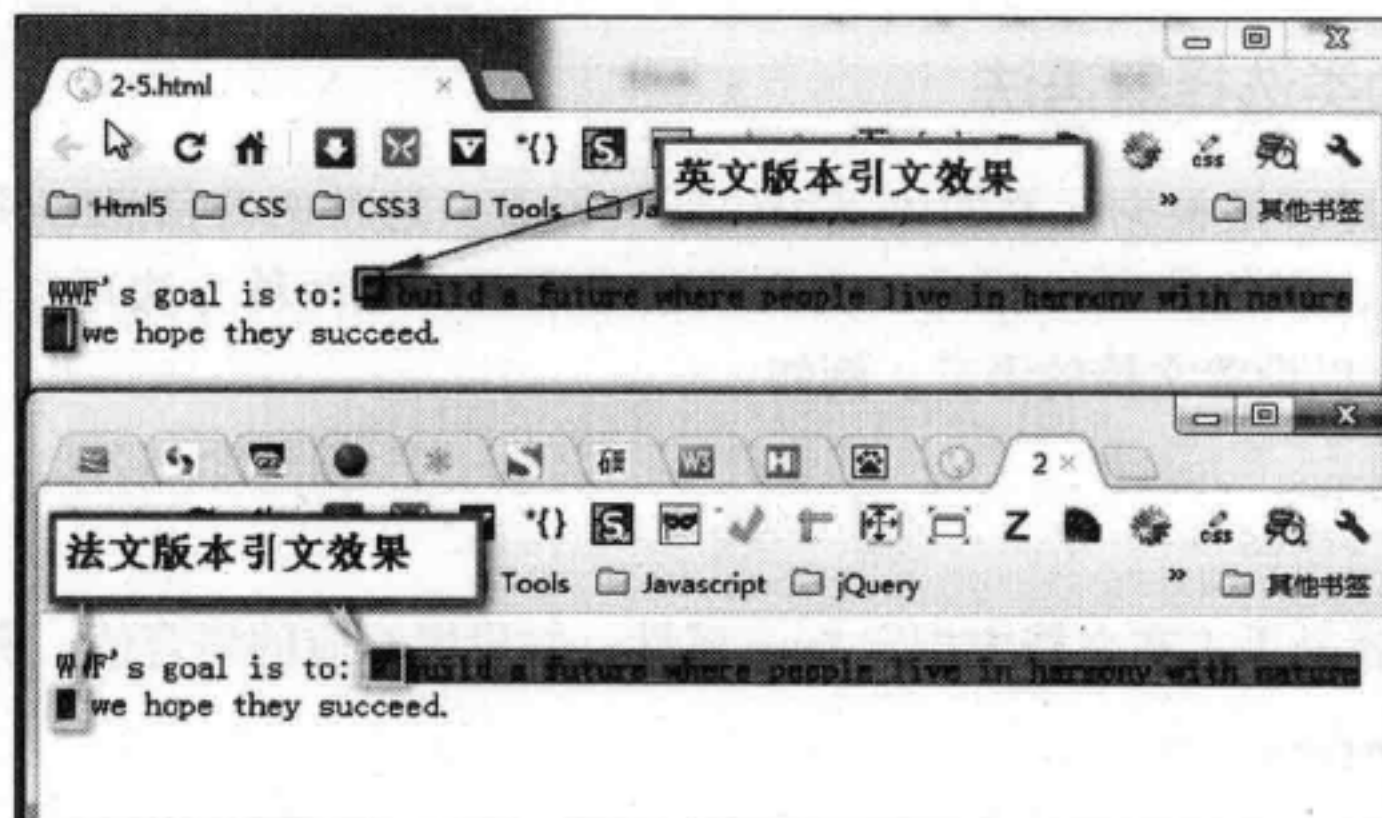
2.6.3 实战体验：定制不同语言版本引文风格

如果网站是一个多语言版本，使用语言伪类选择器为特定的语言定义不同样式是非常完美的。例如，多语言版本有一段这样的引文，如图 2-18 所示。

WWF's goal is to: " build a future where people live in harmony with nature " we hope they succeed.

图 2-18 引文示例

在多语言的网站中，改变引文的不同样式，例如网站还有一个法语语言版本，使用 `« ... »` 替代是不是比引号（`" ... "`）更适合其语言版本呢？同时，为了突出引文的重要性，可以在不同的语言版本中给引文设置不同的背景颜色。最后的效果如图 2-19 所示。



☆图 2-19 多语言版本引文的效果

在不增加任何代码或手工修改代码达到图 2-19 所示的引文效果，使用语言伪类选择器是一个不错的方法，示例代码如下所示。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title> 语言伪类选择器运用 </title>
</head>
<body>
<p>WWF's goal is to:
<q cite="http://www.wwf.org">
build a future where people live in harmony with nature
</q> we hope they succeed.</p>
</body>
</html>
```

为 <html> 标签设置一个 “lang=en-US” 属性，这是默认英文版本的时候。当网站转译到法语版本，此时 <html> 标签中的 lang 属性值动态变成 “fr”，如下所示。

```
<!DOCTYPE HTML>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <title> 语言伪类选择器运用 </title>
</head>
<body>
<p>WWF's goal is to:
<q cite="http://www.wwf.org">
build a future where people live in harmony with nature
</q> we hope they succeed.</p>
</body>
</html>
```

也可以简单地通过目标伪类来实现。

```
/* 英文 (en-US) 版本的引文 q 效果 */
:lang(en) {
  quotes: '""' '""';
}
:lang(en) q {background: red;}
/* 法文 (fr) 版本的引文 q 效果 */
:lang(fr) {
  quotes: '«' '»';
}
:lang(fr) q {background: green;}
```



提示

大家也可以通过这种方法为不同语言版本的网站相关元素设置不同的样式，例如改变网站页面的字号、设置不同的背景图片等。

2.7 UI 元素状态伪类选择器

UI 元素状态伪类选择器也是 CSS3 选择器模块组中的一部分，主要用于 form 表单元素上，以提高网页的人机交互、操作逻辑以及页面的整体美观，使表单页面更具个性与品位，而且使用户操作页面表单更便利和简单。

2.7.1 UI 元素状态伪类选择器语法

UI 元素的状态一般包括：启用、禁用、选中、未选中、获得焦点、失去焦点、锁定和待机。在 HTML 元素中有可用和不可用状态，例如表单中的文本输入框；HTML 元素中还有选中和未选中状态，例如表单中的复选按钮和单选按钮。这几种状态都是 CSS3 选择器中常用的状态伪类选择器，详细说明如表 2-11 所示。

表 2-11 UI 元素状态伪选择器语法






选择器	类型	功能描述
E:checked	选中状态伪类选择器	匹配选中的复选按钮或单选按钮表单元素
E:enabled	启用状态伪类选择器	匹配所有启用的表单元素
E:disabled	不可用状态伪类选择器	匹配所有禁用的表单元素

2.7.2 浏览器兼容性

浏览器兼容性如表 2-12 所示。

除了 IE 浏览器外，各主流浏览器对 UI 元素状态选择器的支持都非常好，但 IE 9

表 2-12 UI 元素状态伪类选择器浏览器兼容性

选择器					
E:checked	9 + √	√	√	√	√
E:enabled	9 + √	√	√	√	√
E:disabled	9 + √	√	√	√	√

也开始全面支持这些 UI 元素状态伪类选择器（如表 2-12 所示）。因此，考虑到 IE 6 ~ 8 是国内用户数最多的浏览器，使用 UI 元素状态伪类选择器需使用特别的方法来处理。

例如使用 JavaScript 库，选用内置已兼容了 UI 元素状态伪类选择器的 JavaScript 库或框架，然后在代码中引入它们并完成想要的效果。由 Keith Clark 编写的 Selectivizr 脚本[Ⓐ]是一个不错的选择。先将该脚本直接引入到页面中，再从 7 个 JavaScript 库中选择一个引入，UI 元素状态伪类选择器就能够在 IE 上工作了。

除了使用 JavaScript 库外，还有一个比较原始而古老的做法，就是在不支持 UI 元素状态伪类选择器的 IE 浏览器下使用类名来处理。例如禁用的按钮效果，可以先在 HTML 标签中添加一个类名“disabled”，就可以在样式中添加样式。

Ⓐ 网址为 <http://selectivizr.com>。


```
.btn.disabled, /* 等效于 .btn:disabled, 用于兼容 IE 低版本浏览器 */
.btn:disabled {
    ...
}
```

2.7.3 实战体验：Bootstrap 的表单元素 UI 状态

UI 元素状态伪类选择器目前主要针对应用在表单元素上，让表单更可用、易用和好用，同时让表单设计更漂亮。这里介绍 Twitter 的表单元素状态是如何来控制的。

Bootstrap 中部分表单元素状态的效果如图 2-20 所示。



图 2-20 Bootstrap 表单元素状态部分效果

图中展示了表单元素中常用的几种 UI 状态效果，接着来看其实现代码。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
    <meta charset="UTF-8">
    <title></title>
    <style type="text/css">
/* 表单基本样式，请查看对应示例代码 */
/* 表单元素获得焦点效果 */
textarea:focus,
input[type="text"]:focus,
input[type="password"]:focus{
    border-color: rgba(82, 168, 236, 0.8);
    outline: 0;
    outline: thin dotted \9;
    box-shadow: inset 0 1px 1px rgba(0, 0, 0, 0.075),
                0 0 8px rgba(82, 168, 236, 0.6);
}
/* 表单中下拉选择框、文件控件、单选按钮、复选按钮得到焦点时效果 */
select:focus,
```

```

input[type="file"]:focus,
input[type="radio"]:focus,
input[type="checkbox"]:focus {
    outline: thin dotted #333;
    outline: 5px auto -webkit-focus-ring-color;
    outline-offset: -2px;
}
/* 禁用的 input、select、textarea 表单元素效果 */
input[disabled], /* 等效于 input:disabled */
select[disabled], /* 等效于 select:disabled */
textarea[disabled] /* 等效于 textarea:disabled */
{
    cursor: not-allowed;
    background-color: #eeeeee;
    border-color: #ddd;
}
/* 禁用的单选按钮和复选按钮效果 */
input[type="radio"][disabled], /* 等效于 input[type="radio"]:disabled */
input[type="checkbox"][disabled] /* 等效于 input[type="checkbox"]:disabled */
{
    background-color: transparent;
}
.control-group.warning > label,
.control-group.warning .help-block,
.control-group.warning .help-inline {
    color: #c09853;
}
/* 表单警告状态下效果 */
.control-group.warning .checkbox,
.control-group.warning .radio,
.control-group.warning input,
.control-group.warning select,
.control-group.warning textarea {
    color: #c09853;
    border-color: #c09853;
}
/* 表单警告状态下并获得焦点下效果 */
.control-group.warning .checkbox:focus,
.control-group.warning .radio:focus,
.control-group.warning input:focus,
.control-group.warning select:focus,
.control-group.warning textarea:focus {
    border-color: #a47e3c;
    box-shadow: 0 0 6px #dbc59e;
}
.control-group.error > label,
.control-group.error .help-block,
.control-group.error .help-inline {
    color: #b94a48;
}
/* 表单错误状态下效果 */

```



```

.control-group.error .checkbox,
.control-group.error .radio,
.control-group.error input,
.control-group.error select,
.control-group.error textarea {
    color: #b94a48;
    border-color: #b94a48;
}
/* 表单错误状态并获取焦点时效果 */
.control-group.error .checkbox:focus,
.control-group.error .radio:focus,
.control-group.error input:focus,
.control-group.error select:focus,
.control-group.error textarea:focus {
    border-color: #953b39;
    box-shadow: 0 0 6px #d59392;
}
.control-group.success > label,
.control-group.success .help-block,
.control-group.success .help-inline {
    color: #468847;
}
/* 表单成功状态下效果 */
.control-group.success .checkbox,
.control-group.success .radio,
.control-group.success input,
.control-group.success select,
.control-group.success textarea {
    color: #468847;
    border-color: #468847;
}
/* 表单成功状态并获取焦点下效果 */
.control-group.success .checkbox:focus,
.control-group.success .radio:focus,
.control-group.success input:focus,
.control-group.success select:focus,
.control-group.success textarea:focus {
    border-color: #356635;
    box-shadow: 0 0 6px #7aba7b;
}
.form-actions {
    padding: 17px 20px 18px;
    margin-top: 18px;
    margin-bottom: 18px;
    background-color: #f5f5f5;
    border-top: 1px solid #e5e5e5;
    *zoom: 1;
}
.form-actions:before,
.form-actions:after {
    display: table;

```

```

    content: "";
}
.form-actions:after {
    clear: both;
}
/* 按钮基本样式 */
.btn {
    display: inline-block;
    *display: inline;
    /* 由于篇幅, 基本样式在此省略, 详细请查阅随书代码 */
}
.btn:hover,
.btn:active,
.btn.active,
.btn.disabled, /* 按钮禁用下效果, 等效于 .btn:disabled */
.btn[disabled] {
    background-color: #e6e6e6;
    *background-color: #d9d9d9;
}
.btn:active,
.btn.active {
    background-color: #cccccc \9;
}

.btn:first-child {
    *margin-left: 0;
}
.btn:hover {
    color: #333333;
    text-decoration: none;
    background-color: #e6e6e6;
    *background-color: #d9d9d9;
    background-position: 0 -15px;
    -webkit-transition: background-position 0.1s linear;
    -moz-transition: background-position 0.1s linear;
    -ms-transition: background-position 0.1s linear;
    -o-transition: background-position 0.1s linear;
    transition: background-position 0.1s linear;
}
.btn:focus {
    outline: thin dotted #333;
    outline: 5px auto -webkit-focus-ring-color;
    outline-offset: -2px;
}
.btn.active,
.btn:active {
    background-color: #e6e6e6;
    background-color: #d9d9d9 \9;
    background-image: none;
    outline: 0;
    box-shadow: inset 0 2px 4px rgba(0, 0, 0, 0.15),

```



```

        0 1px 2px rgba(0, 0, 0, 0.05);
    }
    /* 表单按钮禁用状态下效果 */
    .btn.disabled, /* 等效于 .btn:disabled */
    .btn[disabled] {
        cursor: default;
        background-color: #e6e6e6;
        background-image: none;
        opacity: 0.65;
        filter: alpha(opacity=65);
        box-shadow: none;
    }
    ...// 省略部分代码

```



注意 以上样式代码摘选自 Bootstrap 中的表单元素样式（详细内容参见 <http://twitter.github.com/bootstrap/base-css.html#forms>）。

上面案例主要展示的是表单元素得到焦点和禁用两种状态使用方法，在使用 UI 状态选择器时特别注意，HTML 结构中要存在这种状态，例如禁用的输入框，需要在 HTML 的对应元素上添加禁用属性。

```

<input class="input-xlarge disabled" id="disabledInput"
    type="text" placeholder="Disabled input here..." disabled="">

```

2.8 结构伪类选择器

伪类可以将一段并不存在的 HTML 当作独立元素来定位，或是找到无法使用其他简单选择器就能定位到的切实存在的元素。因此 CSS3 给伪类选择器引入一种“结构伪类选择器”。这种选择器可以根据元素在文档树中的某些特性（如相对位置）定位到它们。也就是说，通过文档树结构的相互关系来匹配特定的元素，从而减少 HTML 文档对 ID 或类名的定义，帮助你保持代码干净和整洁。

2.8.1 重温 HTML 的 DOM 树

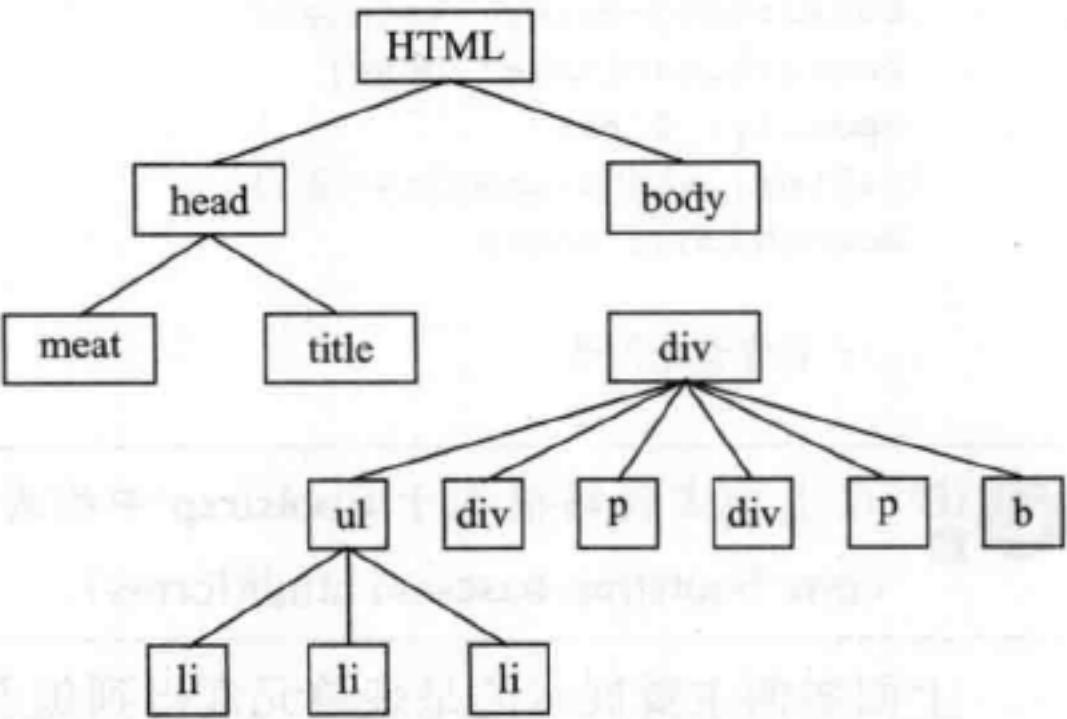
所有的结构伪类都是基于 HTML 文档树的，也称做文档对象模型（DOM），下面简单回顾一下这方面的知识。文档树（Document Tree）是 HTML 页面的层级结构。它由元素、属性和文本组成，它们都是一个节点（Node），就像公司的组织结构图一样。下面看一个简单的 HTML 文档。

```

<!DOCTYPE HTML>
<html lang="en-US">
<head>
    <meta charset="UTF-8">

```

```
<title>HTML DOM 树型结构图 </title>
</head>
<body>
  <div>
    <ul>
      <li>one</li>
      <li>two</li>
      <li>three</li>
    </ul>
    <div>abc</div>
    <p>para</p>
    <div>def</div>
    <p>para</p>
    <b>ghi</b>
  </div>
</body>
</html>
```



依据上面的 HTML 文档，可以绘制一个清晰的 DOM 结构树，如图 2-21 所示。

图 2-21 HTML DOM 树型结构

图 2-21 所示的文档树包含有多个层级是因为 HTML 元素间的相互嵌套。其中把直接嵌入其他元素的元素被称做那些元素的子元素（Child），例如结构图中的 li 就是 ul 的子元素；而随着嵌套的继续深入，它们也就成了后代元素（Descendant），同样，结构图中的 li 元素就是 body 元素的后代元素。那些外部元素称为父元素（Parent）（一层之上），例如结构图中的 ul 元素就是 li 元素的父元素；有些外部元素称做祖先元素（Ancestor）（两层或以上），例如结构图中的 body 就是 li 元素的祖先元素。另外位于相同嵌套层级的元素称为兄弟元素（具有同一父元素节点），例如结构图中的两个段落 P 元素就是兄弟元素，因为它们具有同一个父元素 div。在 HTML 文档中，一个元素可以同时拥有以上部分甚至所有称谓，正如家谱中的某个成员一样，总的来说这些称谓都是用来描述一个元素与另一个元素的关系。

2.8.2 结构伪类选择器语法

通过回顾 HTML 的 DOM 树型结构，清楚了元素之间的关系术语，现在看看有哪些可以建立元素间关系的方法，表 2-13 列出所有结构伪选择器的详细说明。

表 2-13 结构伪类选择器使用语法

选择器	功能描述
E:first-child	作为父元素的第一个子元素的元素 E。与 E:nth-child(1) 等同
E:last-child	作为父元素的最后一个子元素的元素 E。与 E:nth-last-child(1) 等同
E:root	选择匹配元素 E 所在文档的根元素。在 HTML 文档中，根元素始终是 html，此时该选择器与 html 类型选择器匹配的内容相同
E F:nth-child(n)	选择父元素 E 的第 n 个子元素 F。其中 n 可以是整数（1、2、3）、关键字（even、odd）、可以是公式（2n+1、-n+5），而且 n 值起始值为 1，而不是 0

(续)

选择器	功能描述
E F:nth-last-child(n)	选择元素 E 的倒数第 n 个子元素 F。此选择器与 E F:nth-child(n) 选择器计算顺序刚好相反, 但使用方法都是一样的, 其中 :nth-last-child(1) 始终匹配的是最后一个元素, 与 :last-child 等同
E:nth-of-type(n)	选择父元素内具有指定类型的第 n 个 E 元素
E:nth-last-of-type(n)	选择父元素内具有指定类型的倒数第 n 个 E 元素
E:first-of-type	选择父元素内具有指定类型的第一个 E 元素, 与 E:nth-of-type(1) 等同
E:last-of-type	选择父元素内具有指定类型的最后一个 E 元素, 与 E:nth-last-of-type(1) 等同
E:only-child	选择父元素只包含一个子元素, 且该子元素匹配 E 元素
E:only-of-type	选择父元素只包含一个同类型的子元素, 且该子元素匹配 E 元素
E:empty	选择没有子元素的元素, 而且该元素也不包含任何文本节点

表 2-13 中, 只有 “:first-child” 属于 CSS2.1, 此外其他的结构伪类选择器都是 CSS3 的新特性, 为我们提供精确定位到元素的新方式。表中只是告诉我们 CSS3 结构伪类选择器的概念性的知识, 有时候看起来不太好理解, 针对图 2-21 的 HTML DOM 树型结构, 将 CSS3 结构伪类选择器结合起来理解, 如图 2-22 所示。

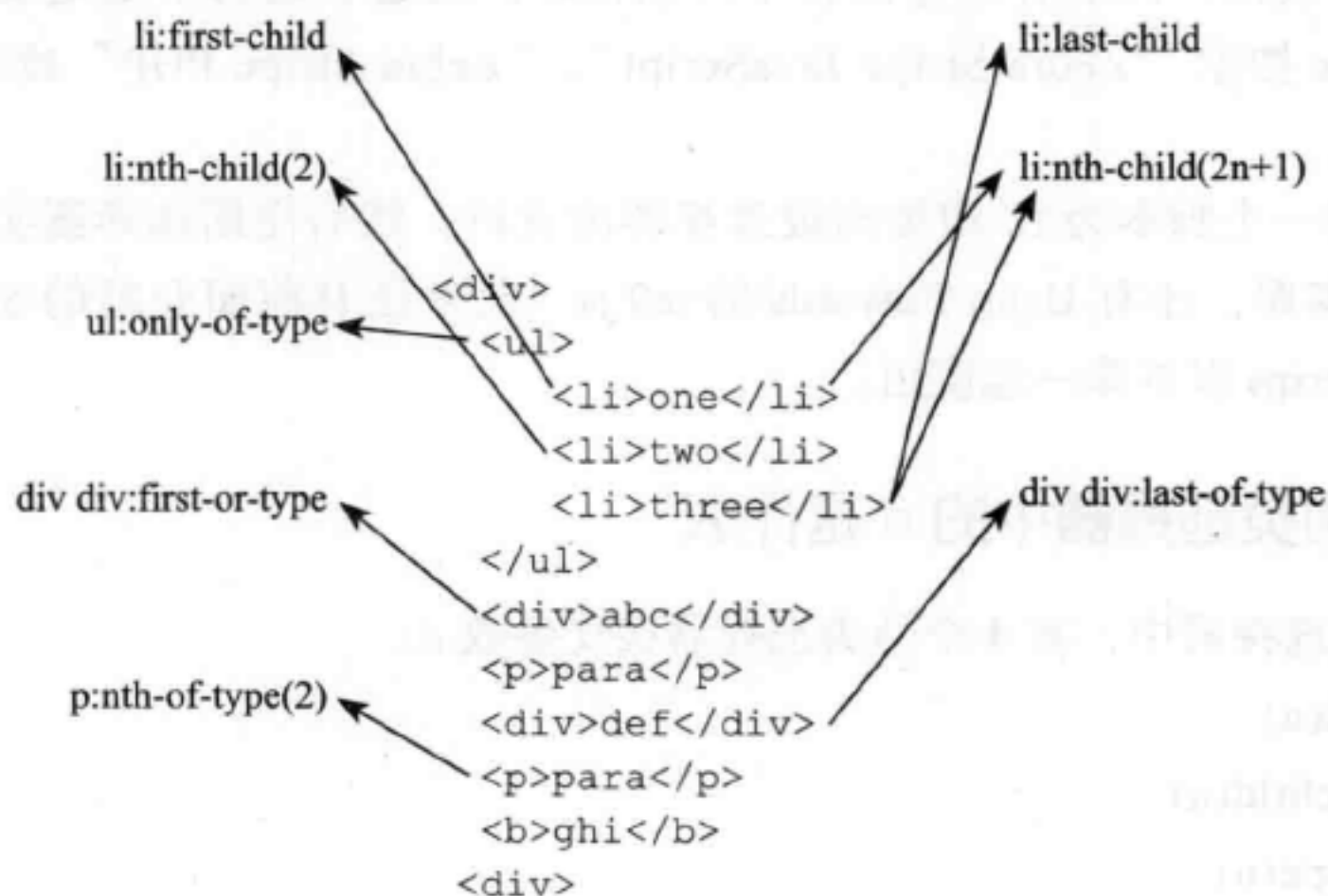


图 2-22 CSS3 结构伪类选择器

2.8.3 浏览器兼容性

CSS3 结构伪类选择器在主流浏览器下运行都非常的完美, 只是在 IE 9 以下版本的浏览器中无法正常运行, 浏览器兼容性如表 2-14 所示。

在本例中颜色交替只是一个很小的视觉增强, IE 8 及之前版本的用户看不到也无妨, 而对于圆角、渐变之类的效果, IE 将直接忽略它们, 使用直角或纯色显示, 看起来没有任何异常之处。

表 2-14 结构伪类选择器浏览器兼容性

选择器					
E:first-child	9 + ✓	✓	✓	✓	✓
E:last-child	9 + ✓	✓	✓	✓	✓
E:root	9 + ✓	✓	✓	✓	✓
E F:nth-child(n)	9 + ✓	✓	✓	✓	✓
E F:nth-last-child(n)	9 + ✓	✓	✓	✓	✓
E:nth-of-type(n)	9 + ✓	✓	✓	✓	✓
E:nth-last-of-type(n)	9 + ✓	✓	✓	✓	✓
E:first-of-type	9 + ✓	✓	✓	✓	✓
E:last-of-type	9 + ✓	✓	✓	✓	✓
E:only-child	9 + ✓	✓	✓	✓	✓
E:only-of-type	9 + ✓	✓	✓	✓	✓
E:empty	9 + ✓	✓	✓	✓	✓

想为 IE 8 及之前的版本提供一个解决方案，需要用到 JavaScript 脚本。例如要实现例中的 Zebra 表格效果，网上有大量的脚本可以帮你。最适合的脚本还是取决于项目本身，可以通过 Google 搜索“Zebra Stripe JavaScript”、“Zebra Stripe PHP”或者其他更符合需求的关键词。

还可以使用一个脚本为 IE 增加高级选择器的支持，然后使用选择器实现需要的效果。例如 jQuery 脚本库，还有 Dean Edwards 的 ie9.js，此外还有前面介绍的 Selectivizr 脚本[⊖]配合其他 JavaScript 脚本库一起使用。

2.8.4 结构伪类选择器中的 n 是什么

在结构伪类选择器中，有 4 个伪类选择器接受参数 n。

- ☐ :nth-child(n)
- ☐ :nth-last-child(n)
- ☐ :nth-of-type(n)
- ☐ :nth-last-of-type(n)

在实际应用中，这个参数 n 可以是整数（1、2、3、4）、关键字（odd、even），还可以是公式（2n+1、-n+5），但参数 n 的起始值始终是 1，而不是 0。换句话说，当参数 n 的值为 0 时，选择器将选择不到任何匹配的元素。

根据上面所述，将结构伪类选择器中的参数按常用的情况划分为七种情形。

1. 参数 n 为具体的数值

这个数值可以是任何大于 0 的正整数，例如“:nth-child(3)”将选择一个系列中的第 3

⊖ 参考 <http://selectivizr.com>。

个元素。

2. 参数 n 为表达式 “ $n \times \text{length}$ ”

选择 n 的倍数，其中 n 从 0 开始计算，而 length 为大于 0 的整数。当 length 为整数 1 时，将选择整个系列中的所有元素，直到元素耗尽无法选择为止。因为 length 在实际运用中常为大于 1 的正数，表达式才具有实际意义，例如 “ $:nth\text{-child}(2n)$ ”。

- $n=0$ 时， $2 \times 0=0$ ，不选中任何元素；
- $n=1$ 时， $2 \times 1=2$ ，选中系列中的第 2 个元素；
- $n=2$ 时， $2 \times 2=4$ ，选中系列中的第 4 个元素；
- $n=3$ 时， $2 \times 3=6$ ，选中系列中的第 6 个元素；
-

以此类推，直到元素耗尽无法选择为止。

3. 参数 n 为表达式 “ $n + \text{length}$ ”

选择大于或等于 length 的元素，例如 “ $:nth\text{-child}(n+3)$ ”。

- $n=0$ 时， $0+3=3$ ，选中系列中的第 3 个元素；
- $n=1$ 时， $1+3=4$ ，选中系列中的第 4 个元素；
- $n=2$ 时， $2+3=5$ ，选中系列中的第 5 个元素；
- $n=3$ 时， $3+3=6$ ，选中系列中的第 6 个元素；
-

以此类推，直到元素耗尽无法选择为止。

4. 参数 n 为表达式 “ $-n + \text{length}$ ”

选择小于或等于 length 的元素，例如 “ $:nth\text{-child}(-n+3)$ ”。

- $n=0$ 时， $-0+3=3$ ，选中系列中的第 3 个元素；
- $n=1$ 时， $-1+3=2$ ，选中系列中的第 2 个元素；
- $n=2$ 时， $-2+3=1$ ，选中系列中的第 1 个元素；
- $n=3$ 时， $-3+3=0$ ，不选择任何元素；
- $n=4$ 时， $-4+3=-1$ ，不选择任何元素；
-

以此类推，当值小于或等于 0 时将不选择任何元素。

5. 参数 n 为表达式 “ $n \times \text{length} + b$ ”

其中 b 是您想设置的偏移值，其表示隔 length 个元素选中第 $n \times \text{length} + b$ 个元素，例如 “ $:nth\text{-child}(2n+1)$ ”。

- $n=0$ 时， $2 \times 0 + 1 = 1$ ，选中系列中的第 1 个元素；
- $n=1$ 时， $2 \times 1 + 1 = 3$ ，选中系列中的第 3 个元素；
- $n=2$ 时， $2 \times 2 + 1 = 5$ ，选中系列中的第 5 个元素；

□ $n=3$ 时, $3 \times 2+1=7$, 选中系列中的第 7 个元素;

.....

以此类推, 直到元素耗尽无法选择为止。

6. 参数 n 为关键词 “odd”

选择系列中的奇数 (1、3、5、7) 元素, 其效果等同于 “:nth-child($2n-1$)” 和 “:nth-child($2n+1$)”。

7. 参数 n 为关键词 “even”

选择系列中的偶数 (2、4、6、8) 元素, 其效果等同于 “:nth-child($2n$)”。Sitepoint.com 也制作了一个关于 “:nth-child(n)” 的参考指南^①供大家对照参考, 如表 2-15 所示。

表 2-15 结构伪类表达式的计算列表

n	$2n+1$	$4n+1$	$4n+4$	$4n$	$5n-2$	$-n+3$
0	1	1	4	—	—	3
1	3	5	8	4	3	2
2	5	9	12	8	8	1
3	7	13	16	12	13	—
4	9	17	20	16	18	—
5	11	21	24	20	23	—

以上几种情形也适用于 “:nth-last-child(n)”、“:nth-of-type(n)” 和 “:nth-last-of-type(n)” 三种结构伪类选择器。

尽管上面公式算出来的数值 (1、3、5、7) 也可以手动给系列元素中的第 1、3、5、7 元素添加对应类名, 但这样做不仅耗时费力, 还容易遗忘, 代码不整洁使维护过程非常痛苦。如果想在已经存在的项目中插入另外一个, 不得不对插入处之后的项目全部重新定义新的类名, 因为插入后新的序号被打乱。而用 CSS3 的结构伪类选择器 “:nth-child(n)” 代替类名, 跟踪元素系列的序号变化并自动匹配将会更为准确、高效, 而且维护非常方便。

或许会觉得这样的数学计算太麻烦, 从而产生对 “nth-child(n)” 系列结构伪类选择器的抵触。大家不用担心, 线上有一些不错的工具, 可以通过更改数值为即时查看它是如何影响页面样式的, 这将有助于更好地了解 “:nth-child(n)” 系列结构伪类选择器的作用, 例如:

□ Lea Verou 制作的工具^②。

□ Chris Coyier 制作的工具^③。

□ Neal Grosskopf 制作的工具^④。

① 网址为 <http://reference.sitepoint.com/css/understandingnthchildexpressions>。

② <http://lea.verou.me/demos/nth.html>, 笔者比较喜欢的一个工具。

③ <http://css-tricks.com/examples/nth-child-tester/>, 比较基础, 只展示了 “:nth-child(n)” 的使用效果。

④ <http://www.nealgrosskopf.com/tech/resources/80/>。

2.8.5 结构伪类选择器的使用方法详解

结构伪类选择器是 CSS3 选择器最具特色的一部分内容，同时也是 CSS3 选择器中最出色的一部分内容。通过前面的内容，大家对 CSS3 的结构伪类选择器有了初步的了解，但是完全理解它们，还是需要一定的实例，下面分别介绍 CSS3 的结构伪类选择器的具体使用方法。

为了更好地实例化，先创建一个简单的列表结构，并附上一些简单的样式。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>CSS3 结构伪选择器的使用 </title>
  <style type="text/css">
    *{
      margin: 0;
      padding: 0;
    }
    ul {
      margin: 50px auto;
      width: 400px;
      list-style: none outside none;
    }
    li {
      display:inline-block;
      margin: 5px;
      padding: 5px;
      width:50px;
      height: 50px;
      font: bold 30px/50px arial;
      background: #000;
      color: #fff;
      border-radius: 50px;
      text-align: center;
    }
  </style>
</head>
<body>
  <ul>
    <li>1</li>
    <li>2</li>
    ...
    <li>20</li>
  </ul>
</body>
</html>
```

页面初始效果如图 2-23 所示。

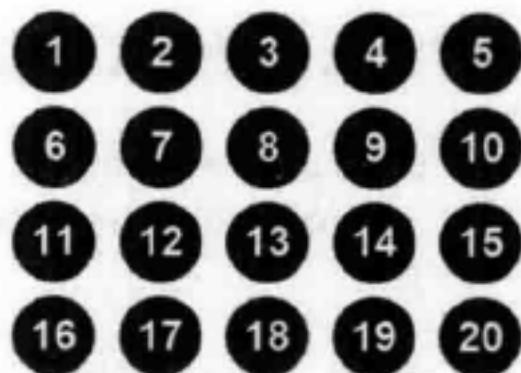


图 2-23 页面初始效果

1. :first-child 的使用

“:first-child”允许定位某个元素的第一个子元素，例如想改变列表中的第一个“li”的

背景色，代码如下。

```
ul>li:first-child {
    background-color: green;
}
```

在没有这个选择器之前，需要在列表中的第一个“li”加上一个类名，例如“first”，然后给添加对应的样式。

```
ul>li.first {
    background-color: green;
}
```

其实这两种最终效果是一样的，如图 2-24 所示。后面这种需要在 html 标签中增加一个额外的 class 类名，只是一个地方还好处理，如果是多处都具有这样的效果，给 html 添加类名的方法其弊端明显可见。

在实际项目中这样的运用也是常有的事，例如博客侧边栏的标题“h2”顶部都有一个“margin”，用来区分标题和它们前面区块的内容，但是第一个标题“h2”不需要顶部“margin”值，就可以使用下面的代码。

```
.aside > h2 {
    margin-top: 15px;
}
.aside>h2:first-child {
    margin-top: 0;
}
```

上面看到的是使用“:first-child”来移除标题顶部的间距，当然也可以用来移除元素底部的间距，此时在不支持“:first-child”的浏览器中，布局并不会因此而破坏掉，它只会看起来有些不同（顶部或底部间距没清除）。但是，如果使用“:first-child”来移除一个浮动元素的左边距或右边距，在不支持“:first-child”的浏览器中，布局将会被破坏掉。

2. :last-child 的使用

“:last-child”与“:first-child”作用类似，不同的是“:last-child”选择的是元素最后一个子元素。就拿上面的例子来说，改变列表中最后一个“li”的背景色，使用这个选择器如下所示。

```
ul>li:last-child {
    background-color: blue;
}
```

和“:first-child”一样，以前为了实现上面的效果，都是在列表中的最后一个“li”添加一个类名“last”，并在此类添加样式，其最终效果都是一样的，如图 2-25 所示。

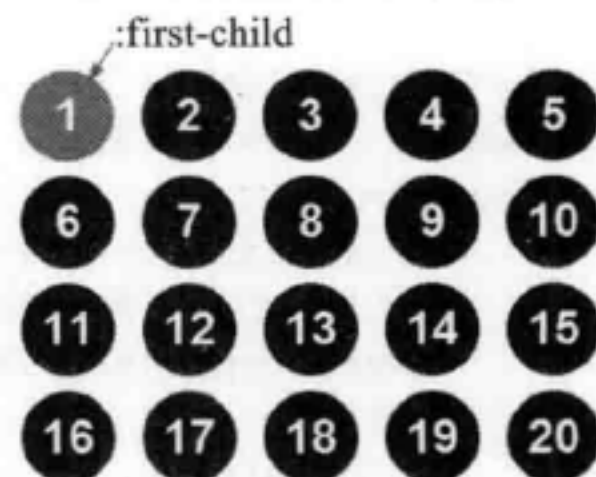


图 2-24 :first-child 效果



图 2-25 :last-child 效果

在实际的 Web 项目中，“:last-child”的用处也非常广泛。例如在一个导航条中每个导航项都有一个右边框效果，但最后一个不想要这个右边框，此时就可以使用“:last-child”。

```
#nav > li {
    ...
    border-right: 1px solid #ccc;
}
#nav > li:last-child {
    border-right: none;
}
```

另一个较常见的就是在博客制作中，假设“post”中最后一段不需要底部“margin”值，代码如下。

```
.post > p:last-child {
    margin-bottom: 0;
}
```

3. :nth-child 的使用

“:nth-child()”用来定位某个父元素的一个或多个特定的子元素。“:nth-child()”可以接受参数 n，而且 n 可以是数值，也可以是表达式和关键词。有关于“n”是什么？大家可以参考“结构伪类选择器中的 n 是什么”。在这一节中，通过实例加深理解“:nth-child(n)”。

(1) :nth-child(3)

参数 n 是一个具体的整数值，例如“:nth-child(3)”表示选择某元素下的第 3 个子元素，(这里的整数 3 可以根据自己的需要来定义)。接上面的实例，如果需要改变列表项中第 3 个“li”元素的背景色，就可以直接这样使用。

```
ul>li:nth-child(3){
    background-color: yellow;
}
```

这样一来，列表中的第 3 个 li 的背景就变成黄色了，如图 2-26 所示。

上面仅是列表中存在 li 一种子元素，但是如果列表中第 3 个 li 之前还有其他的子元素，例如 DIV 元素（当然这样写 HTML 是一种不规范的写法，此处仅用来说明问题，不提倡这样去写 HTML 结构），“ul>li:nth-child(3)”还会选中列表中的第 3 个 li 元素吗？先不做任何回答，改变一下实例的代码。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
    <meta charset="UTF-8">
    <title></title>
```

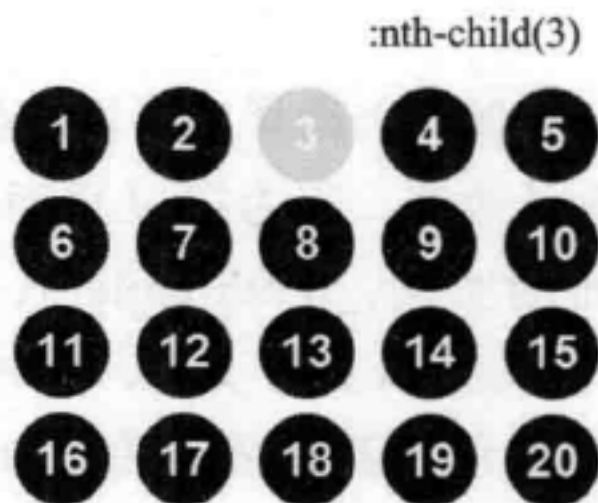


图 2-26 :nth-child(3) 效果

```

<style type="text/css">
  ul {
    list-style: none outside none;
    padding: 10px;
    background: green;
    width: 400px;
  }
  li {margin-bottom: 10px;border: 1px solid orange;}
  div {margin-bottom: 10px;border: 1px solid blue;}
</style>
</head>
<body>
  <ul>
    <li>1</li>
    <li>2</li>
    <div>div</div>
    <div>div</div>
    <li>3</li>
    ...
    <li>10</li>
  </ul>
</body>
</html>

```

页面的初步效果如图 2-27 所示。

同样，加上以下代码。

```

ul>li:nth-child(3){
  background-color: yellow;
}

```

保存上面样式后刷新浏览器，并没有得到想要的效果，列表中的第 3 个 li 背景色没有为此改变。因为在 ul 里面有其他类型的元素（不是 li），它也会算作是列表的子元素。就上面的实例，li:nth-child(3) 并不存在，列表的第 3 个子元素是 div 而不是 li，此时如果还想改变第 3 个列表的背景色，就必须改变“:nth-child()”中的值（或者使用 :nth-of-type()）。拿这个实例来说，第 3 个 li 是列表中的第 5 个子元素，将上面的代码改为以下形式。

```

ul>li:nth-child(5){
  background-color:yellow;
}

```

接下来用效果来证明是不是需要的效果，如图 2-28 所示。

通过上面两个实例的对比，大家清楚“ul>li:nth-child(3)”表达的并不是一定选择列表 ul 元素中的第 3 个

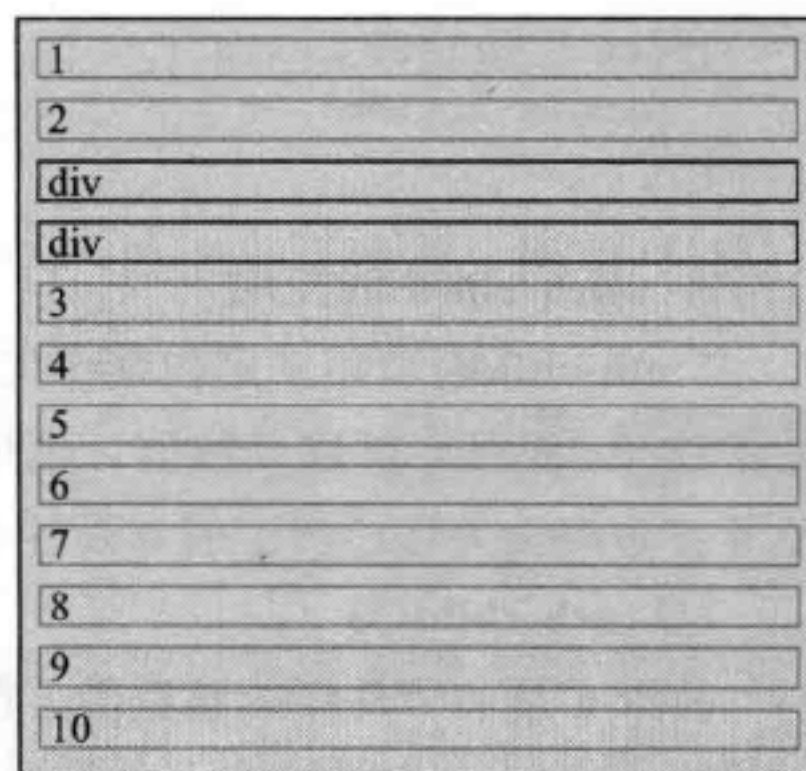


图 2-27 页面初步效果

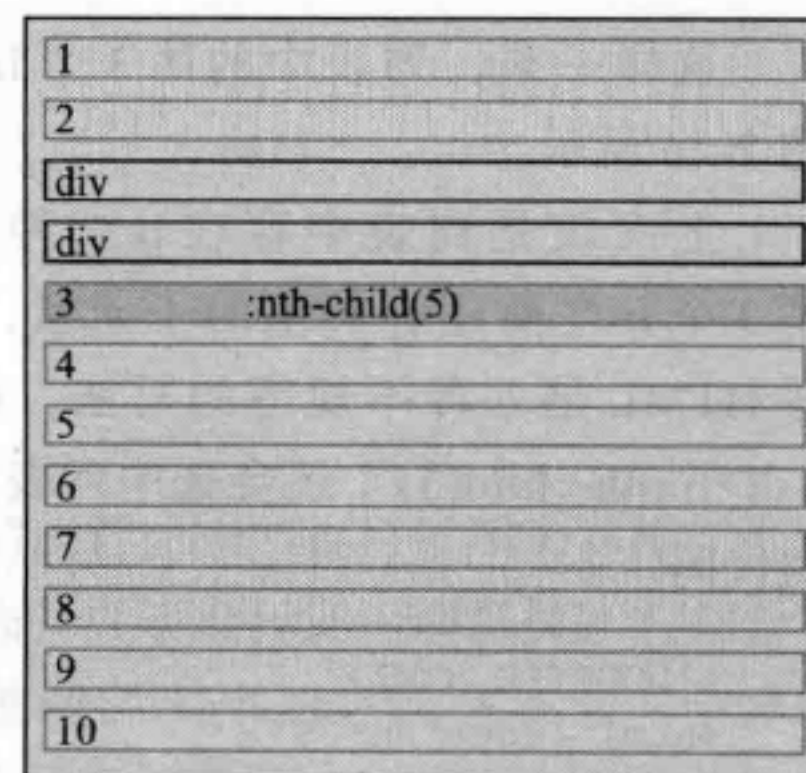


图 2-28 :nth-child(5) 改变第 3 个 li 背景效果

子元素 li，仅有列表 ul 中第 3 个 li 元素前不存在其他的元素，命题才有意义，否则就会造成开始的问题，不会改变列表第 3 个 li 元素的背景色。

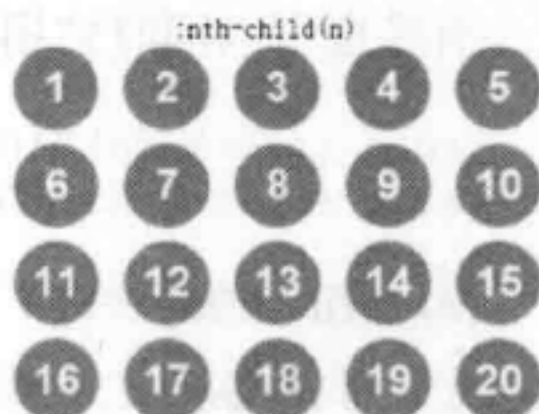
(2) :nth-child(n)

参数 n 是一个简单的表达式，取值从 0 开始计算的，到什么时候结束不知道，如果在实际应用中直接这样使用，将会选中父元素中的所有子元素。

接着前面列表的案例，在代码中加入以下代码。

```
ul>li:nth-child(n){
    background-color: orange;
}
```

效果如图 2-29 所示。



☆图 2-29 :nth-child(n) 效果



注意 “:nth-child(n)” 中参数只能是 n，不可以使用其他的字母代替，不然会没有任何效果。

(3) :nth-child(2n)

该选择器是 “:nth-child(n)” 的一种变身，可以选择 n 的 2 倍数，当然其中 “2” 可以换成需要的数字。

```
ul>li:nth-child(2n){
    background-color: blue;
}
```

这样列表中的偶数项都将被选中，其效果如图 2-30 所示。

这个时候大家有没有想到关键词 “even” 呢？是的，“:nth-child(2n)” 和使用关键词 “:nth-child(even)” 得到的效果是一样的。它们在实际项目中用来制作一些突出效果是非常方便的。例如制作斑马线表格，给偶数行设置一个不同的背景色。



☆图 2-30 :nth-child(2n) 效果

```
table tr:nth-child(even){...}
```

或者：

```
table tr:nth-child(2n){...}
```

(4) :nth-child(2n+1)

该选择器是在 “:nth-child(2n)” 基础上演变过来的，前面说了 “:nth-child(2n)” 和 “:nth-child(even)” 是选择偶数，就可以在其基础上加 1 或减 1，将偶数变成奇数。例如：

```
ul>li:nth-child(2n+1){
    background-color: blue;
}
```

这样列表中奇数列背景色都将变成蓝色，如图 2-31 所示。

“:nth-child(2n+1)”和“:nth-child(2n-1)”选择的是父元素中排序为奇数的子元素，同时选择奇数还可以直接使用关键词“odd”来实现（“:nth-child(odd)”）。换句话说，“:nth-child(odd)”、“:nth-child(2n+1)”和“:nth-child(2n-1)”三个选择器定位的元素是完全相同的。

使用奇数选择器制作斑马线表格是很常见的，不同之处是改变的是表格奇数行的背景色，当然也可以同时使用偶数和奇数，制作一个靓丽的斑马线表格。

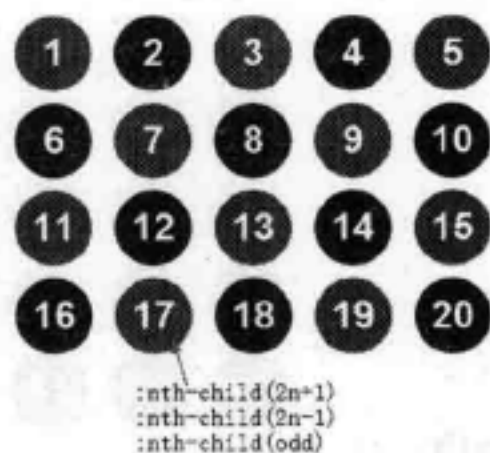
```
table tr:nth-child(odd) {...}/* 设置表格奇数行样式 */
table tr:nth-child(even){...}/* 设置表格偶数行样式 */
```

(5) :nth-child(n+5)

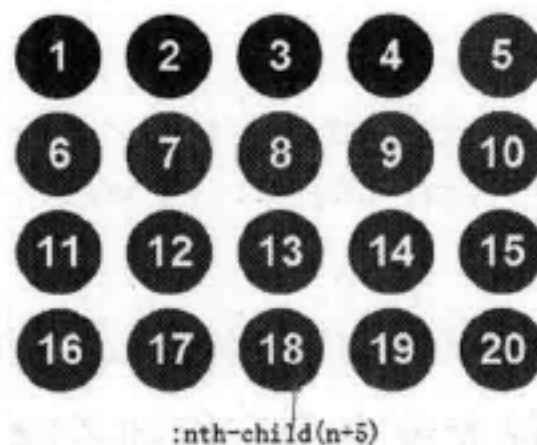
这个选择器从父元素中的第5个子元素开始选择，这里的数字可以自己定义。例如：

```
ul>li:nth-child(n+5){
    background-color: blue;
}
```

此时列表中第5个li元素开始直到最后一个li元素，其背景都将变成蓝色，如图2-32所示。



☆图 2-31 :nth-child(2n+1) 效果



☆图 2-32 :nth-child(n+5) 效果

例如在博文有一个“今日焦点”区块，想让今日点击量落后于第三的文章序列号标注红色，此时这个选择器就非常的方便了，如图2-33所示。

只要将选择器中的数字变成需要的即可。

```
.post-hot-today li:nth-child(n+4){color:
red;}
```

(6) :nth-child(-n+5)

该选择器刚好和“:nth-child(n+5)”相反，选择父元素中第1个到第5个子元素，如果不太清楚，只要把两个表达式的计算值对比就非常清楚了。同样通过“:nth-child(-n+5)”来选择列表中前5个子元素li。

今日焦点

01. CSS3实现11种经典的CSS技术 (34)
02. 浏览器兼容之旅的第二站:各浏览器的Hack写法 (7)
03. 修复iPhone上submit按钮bug (6)
04. CSS——Bootstrap From Twitter (4)
05. CSS3制作超酷的SearchBox (4)
06. CSS3 Transform (3)
07. CSS3的REM设置字体大小 (3)
08. notepad++结合Zen Coding快速编写HTML代码 (3)
09. 表单button的行高问题 (2)
10. CSS3 Multiple Backgrounds (2)

更多

图 2-33 点击量未进前三的序列号标注红色


```
ul>li:nth-child(-n+5){
    background-color: blue;
}
```

上面代码刚好与图 2-32 效果相反,如图 2-34 所示。

同样,想在博客“热门博文”区块内把排在前三的文章进行特殊标注,如图 2-35 所示。

选择器的使用如下所示。

```
.post-hot li:nth-child(-n+3){...}
```

(7) :nth-child(4n+1)

选择器实现某父元素的子元素隔几选一的效果,例如“:nth-child(4n+1)”实现的就是隔 3 个子元素选中一个子元素,直到所有元素耗尽为止。其中“4n+1”可以根据自己的需求进行修改。在前面实例基础上,使用“:nth-child(4n+1)”来改变 li 的效果色为蓝色。

```
ul > li:nth-child(4n+1){
    background-color: blue;
}
```

此时整个列表项的第 1、5、9、13、17 项背景色变成蓝色,如图 2-36 所示。

上面展示的是“:nth-child”结构伪类选择器的几种使用方法,大家在实际应用中根据需求使用不同的表达式,从而满足需求。

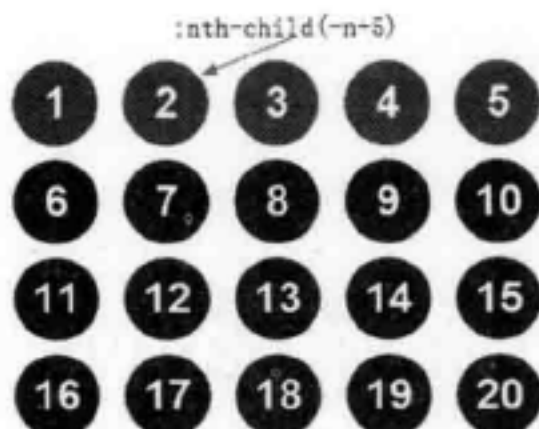
4. :nth-last-child 的使用

“:nth-last-child”和“:nth-child”相似,只是这里多了一个“last”,作用和“:nth-child”有所区别。从某父元素的最最后一个子元素开始计算来选择特定的元素。接着上面的实例代码,来看“:nth-last-child”选择器的实例。

```
ul>li:nth-last-child(4){
    background-color: blue;
}
```

上面的代码选择了 ul 列表中倒数第 4 个 li 元素,如图 2-37 所示。

“:nth-last-child”可以像“:nth-child”一样使用表达式、关键词等,不同的是,“:nth-last-child”起始位置是从父元素的最后一个子元素开始计算。注意“:nth-last-child”使用关



☆图 2-34 :nth-child(-n+5) 效果

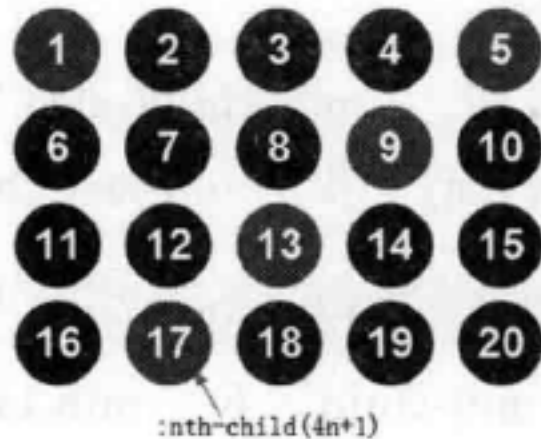
热门博文

:nth-child(-n+3)

01. CSS3实现11种经典的CSS技术 (15,161)
02. CSS3的文字阴影—text-shadow (10,244)
03. CSS3的圆角Border-radius (9,811)
04. CSS3 Gradient (9,516)
05. CSS3 Transition (9,400)
06. CSS—Bootstrap From Twitter (9,251)
07. CSS3 box-shadow (8,376)
08. CSS3 Transform (7,219)
09. notepad++结合Zen Coding快速编写HTML代码 (6,976)
10. CSS3 Animation (6,889)

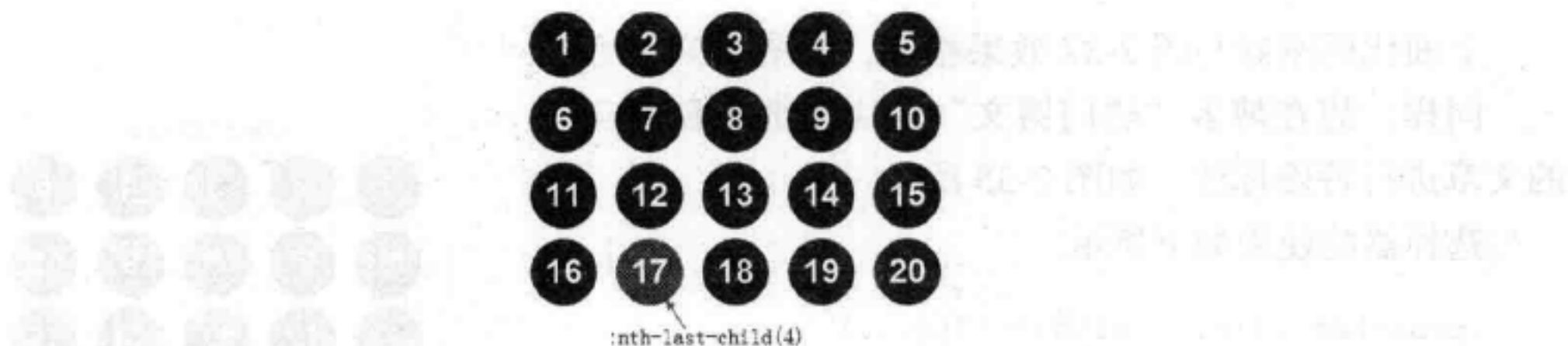
更多

图 2-35 热点博文排在前三的特殊效果



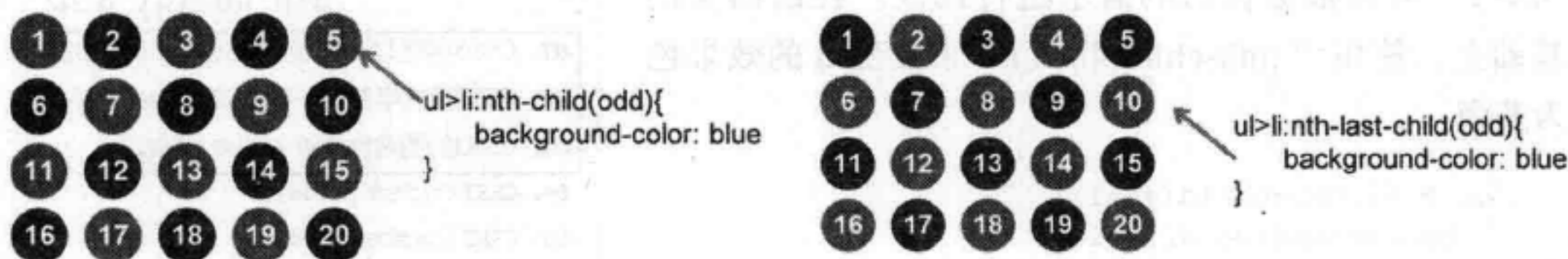
☆图 2-36 :nth-child(4n+1) 效果

关键词 odd、even，由于起始位置不同，“:nth-last-child”使用关键词 odd 和 even 所得到的顺序刚好相反。

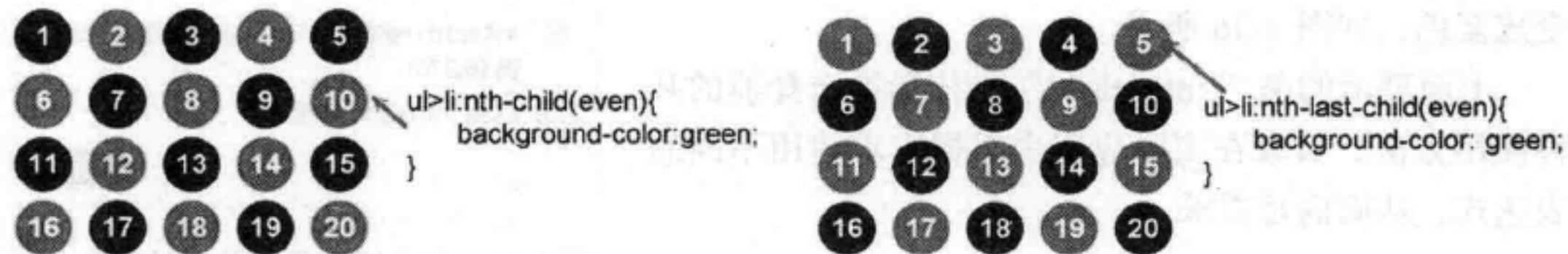


☆图 2-37 :nth-last-child(4) 效果

简单来说，使用“:nth-child(odd)”选择的是奇数项，而使用“:nth-last-child(odd)”选择的却是偶数项，如图 2-38 所示。同样，使用“:nth-child(even)”选择的是偶数项，而使用“:nth-last-child(even)”选择的是奇数项，如图 2-39 所示。



☆图 2-38 :nth-child(odd) 与 :nth-last-child(odd) 对比



☆图 2-39 :nth-child(even) 和 :nth-last-child(even) 对比

可知，“:nth-child(odd)”与“:nth-last-child(even)”选择的元素是相同的，而“:nth-child(even)”和“:nth-last-child(odd)”选择的元素相同。

扩展阅读 “:nth-child”与“:nth-last-child”的区别

“:nth-child”和“:nth-last-child”两个选择器的使用方法和所起的作用是一样的，用来选择某父元素中的特定子元素，同时所有的子元素不分类型，而且所有出现的子元素都会按文档流的先后顺序来排序。同时它们之间有一个很明显的区别。

- “:nth-child”选择器选择的子元素是从第一个子元素开始算起；
- “:nth-last-child”选择器选择的子元素却是从最后一个子元素开始算起。

5. :nth-of-type 的使用

“:nth-of-type”和“:nth-child”类似，不同的是它只计算父元素中指定的某种类型的子元素。当某个元素中的子元素不单单是同一种类型的子元素这，使用“:nth-of-type”选择器来定位于父元素中某种类型的子元素是非常的方便和有用。正如前面看到的实例，当ul列表中不仅仅有子元素li，而且还有别的子元素是DIV，使用ul>li:nth-child(3)来选择第3个li元素时，无法选择，如图2-27所示。通过改变选择器“:nth-child”的变量参数值为5，才可以选中列表中的第3个子元素li，如图2-28所示。但是使用“:nth-of-type”，就不用改变其变量参数值。

```
ul>li:nth-of-type(3){
    background-color:orange;
}
```

效果如图2-40所示。

同“:nth-child”的具体使用方法一样，“:nth-of-type”也具有参数设置，这个参数也是n，它可以是具体的整数值，也可以是表达式和关键词。

在Web应用中，“:nth-of-type”在以下场景中使用。

- 营造一种有随意感的界面，例如改变每张图片的旋转角度；
- 使文章中的图片交替着向左向右浮动；
- 为一篇文章的头一段设置不同的样式，例如首字下沉；
- 为一个定义列表的条上使用交替样式；
- 制作图表。

此外，还有更多的使用场景。一起来看看其中几个场景下的具体使用。

在Web页面的设计中，常常给某种类型元素设置一些特殊的样式，以达到突出的目的。例如，将一篇文章第一个段落的文字设置大一些，以前通常是在第一个段落p添加一个类名“first”，然后设置其字号比别的段落大一些。使用“:nth-of-type”选择器，就能轻松的改变文章中第一个段落的字号。

```
.node p:nth-of-type(1){
    font-size: 1.5em;
}
```

也许想在一篇文章中把奇数图片左对齐，偶数图片右对齐，如图2-41所示。

首先想到给文章中的图片加一个left或right类名，然后分别给这两个不同的类名定义不同的对齐方式。这样的做法毫无疑问是正确的，但现在说最佳的做法是使用“:nth-of-type”来实现。

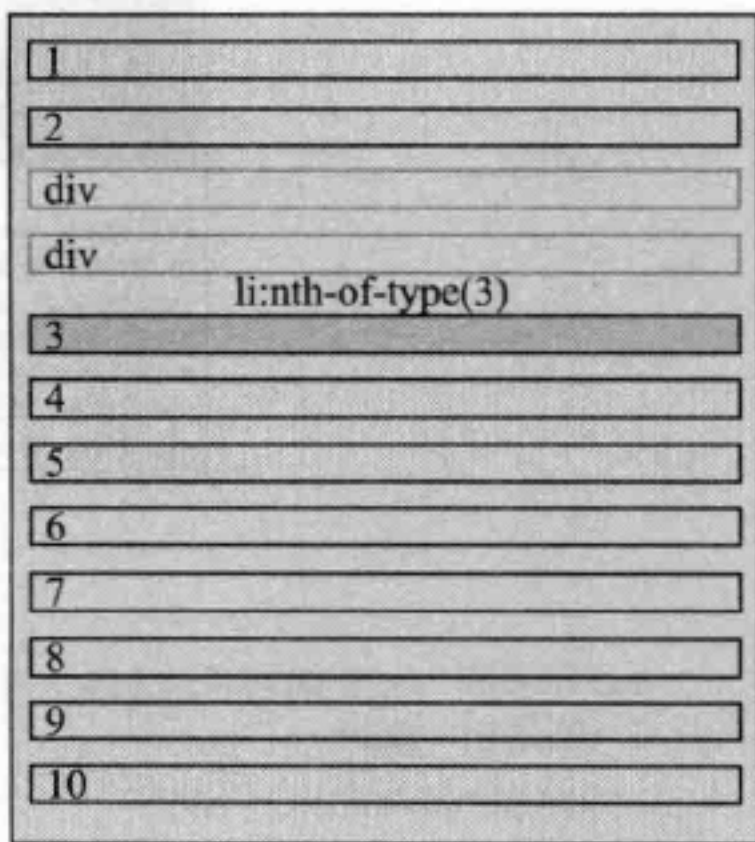


图2-40 :nth-of-type(3) 效果



图 2-41 图片对齐效果

```
.article img:nth-of-type(odd) {
  float: left;
  margin-right: 10px;
}
.article img:nth-of-type(even) {
  float: right;
  margin-left: 10px;
}
```

它在 IE 8 及以下浏览器是不被支持的，但可以借助 JavaScript 库来实现，例如 Keith Clark 编写的 Selectivizr 脚本^①，或者直接无视不支持的浏览器，例如 Simon Foster^②就使用了“:nth-of-type”为它收集的 45RPM 唱片制作的一份漂亮的图表，采用“:nth-of-type”为每种不同的流派设置一个不同的背景图片。下面是 Simon Foster 站上截取的一段代码。

```
ul#genre li:nth-of-type(1) {
  width:32.9%;
  background:url(images/orangenoise.jpg);
}
ul#genre li:nth-of-type(2) {
  width:15.2%;
  background:url(images/bluenoise.jpg);
}
ul#genre li:nth-of-type(3) {
  width:13.1%;
```

① 详见 <http://selectivizr.com>。

② 详见 <http://www.fortherecord.simonfosterdesign.com/>。


```
background:url(images/greennoise.jpg);
}
```

图 2-42 是从站点截取的效果。

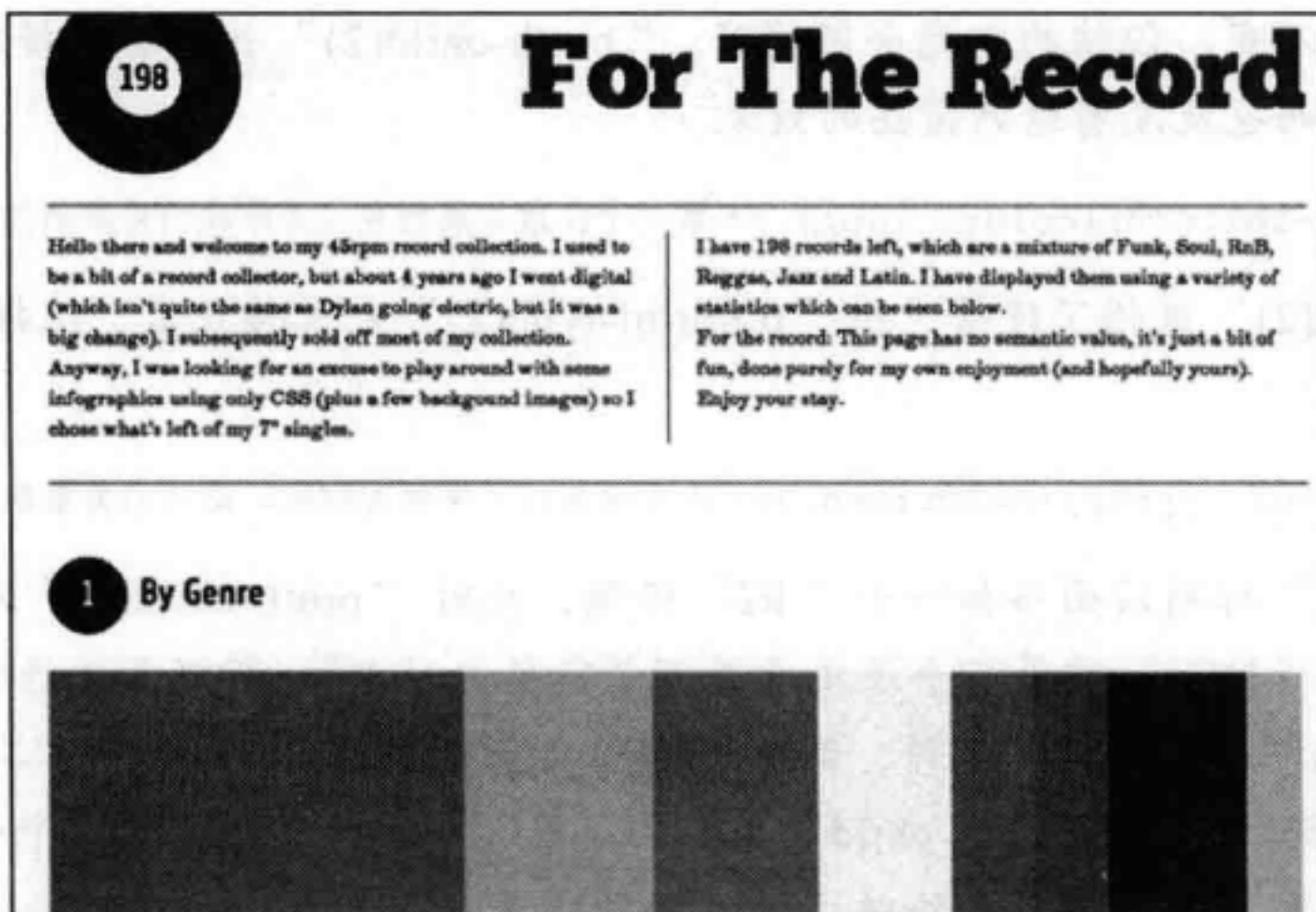


图 2-42 “For The Record” 站点使用 :nth-of-type 的效果

扩展阅读 “:nth-child” 和 “:nth-of-type” 的区别

“:nth-child” 和 “:nth-of-type” 两者之间的区别对于初学者来说很是头痛的问题，为了更好地帮助大家区分使用方法，特意在此加以区分。首先创建一个简单的 HTML 结构。

```
<div class="post">
  <p> 我是文章中的第一个段落 </p>
  <p> 我是文章中的第二个段落 </p><!-- == 我要变成红色的字 == -->
</div>
```

接下来，使用 “:nth-child” 和 “:nth-last-child” 选择段落并改变其文字颜色。

```
.post>p:nth-child(2){color:red;}
.post>p:nth-of-type(2){color:red;}
```

上面的代码都把 “post” 中的第二段文字变成大红色，是不是代表这两个选择器就是一样呢？其实不然。“:nth-child” 仅从字面上来解释，其包含了两层意思。首先是一个段落元素，而且这个段落是父元素 “DIV” 的第二个子元素；而 “:nth-of-type” 从字面上解释是 “选择父元素 DIV 的段落二”。

上面一段话看来很晕，有没有更好的方法来区分它们呢？有的，把上面的 HTML 结构改变一下，在两个段落前加上一个标题 “h1”。

```
<div class="post">
```

```

<h1>我是标题一</h1>
<p>我是文章中的第一个段落</p>
<p>我是文章中的第二个段落</p> <!-- == 我要变成红色的字 == -->
</div>

```

前面的样式不变，但结构却完全不同了。“p:nth-child(2)”并没有选择段落二，而是选择了段落一，从而也就没有达到需要的效果。

```
.post>p:nth-child(2){color:red;} /* 第一个段落变成红色，不是我们需要的效果 */
```

“p:nth-child(2)”选错了段落，但“p:nth-of-type(2)”却工作正常，选择的还是段落二，实现想要的效果。

```
.post>p:nth-of-type(2){color:red;} /* 改变段落二文字色为红色，是我们需要的效果 */
```

如果在“h1”标题后面添加一个“h2”标题，此时“p:nth-child(2)”将无法选择任何元素，因为此时“DIV”的第二个子元素并不是段落一“P”，所以无法选择任何元素。但“p:nth-of-type(2)”依然能正常工作，因为选择的始终是“DIV”中的段落二“P”。

大家只需要记住一点：“:nth-child”选择的是某父元素的子元素，这个子元素并没有指定确切的类型，同时满足两个条件时方能有效果，其一是子元素，其二此子元素刚好处在那个位置；“:nth-of-type”选择的是某父元素的子元素，而且这个子元素是指定类型。

“:nth-child”虽然常见，但却脆弱，正如前面的示例所示，随时被其他子元素给挤出选择的范围。而“:nth-of-type”不常见，但在选择某种类型的子元素时，更稳定，更可靠。

6. :nth-last-of-type 的使用

“:nth-last-of-type”和“:nth-of-type”一样，用来选择父元素中指定的某种子元素类型，但它的起始方向是从最后一个子元素开始，而且使用方法可以像前面提到的“:nth-last-child”一样使用。

例如选择一篇文章的最后一个段落，会很快地想到“:last-child”和“:nth-last-child(1)”这两个选择器，但是文章中并不常常都是以段落来结束，这个时候“:nth-last-of-type”就能快速、准确定位到最后一个段落。

```
.article p:nth-last-of-type(1){...}
```

也可以更加的聪明一些，在一个块级选择器中结合多种这样的伪类选择器。举个例子，让文章中除了第一个和最后一个的所有图片左浮动，这个时候多种伪类的选择器就非常的实用。

```
.article img:nth-of-type(n+2):nth-last-of-type(n+2){
  float:left
}
```

在这个组合型选择器中，第一部分“:nth-of-type(n+2)”从第2个图片开始定位，选择了文章中所有图片，除了第一张；第二部分“:nth-last-of-type(n+2)”从文章中倒数第2张

图片开始定位, 选择了文章中所有图片, 除了最后一张。因为这两个选择器并非互相排斥的, 可以同时使用它们, 这样就可以立刻排除第一张和最后一张图片, 达到想要的选择效果。

扩展阅读 “:nth-of-type” 和 “:nth-last-of-type” 的区别

“:nth-of-type” 和 “:nth-last-of-type” 都是结构伪类选择器, 它们的作用也是一样的, 都是用来选择某父元素中指定类型的子元素, 区别就是, “:nth-of-type” 选择的某类型子元素是从前往后排序计算, 而 “:nth-last-of-type” 选择的某类型子元素是从后向前排序计算。

7. :first-of-type 和 :last-of-type 的使用

“:first-of-type” 和 “:last-of-type” 这两个选择器类似于 “:first-child” 和 “:last-child”, 不同之处就是指定了元素的类型。换句话说, “:first-of-type” 是用来定位一个父元素下的某个类型的第一个子元素; 而 “:last-of-type” 用来定位一个父元素下的某个类型的最后一个子元素。

通过前面的学习了解到, “:nth-of-type(1)” 用来选择父元素指定类型第一个元素, 其实可以使用 “:first-of-type” 来代替。同样, 可以使用 “:last-of-type” 来代替 “:nth-last-of-type(1)”。

8. :only-child 的使用

“:only-child” 表示一个元素是它父元素的唯一子元素。换句话说, 匹配元素的父元素中仅有一个子元素。来看一个简单的例子, 在 post 列表中, 有的只有一个段落, 有的不只一个段落。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>:only-child 的使用 </title>
  <style type="text/css">
    .post {
      width: 300px;
      margin: 20px auto;
      padding: 5px;
      border: 1px solid #ccc;
    }
    p {
      background: green;
      color: #fff;
      border: 1px solid orange;
      padding: 5px;
    }
  </style>
</head>
```

```

<body>
  <div class="post">
    <p> 我是第一个段落 </p>
    <p> 我是第二个段落 </p>
  </div>
  <div class="post">
    <p> 我就一个段落 </p>
  </div>
</body>
</html>

```

上面的实例中运用了一些初步样式，其初步页面效果如图 2-43 所示。

下面是关键时候了，使用“:only-child”看看两个 post 中的段落 p 会有什么样的变化。

```

.post>p:only-child {
  border-width:2px;
  background-color:#000;
}

```

使用“:only-child”后的效果如图 2-44 所示。

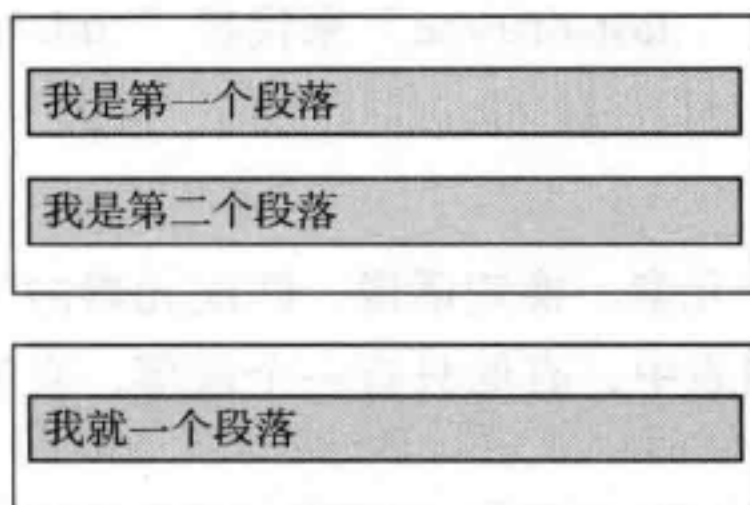


图 2-43 页面初步效果

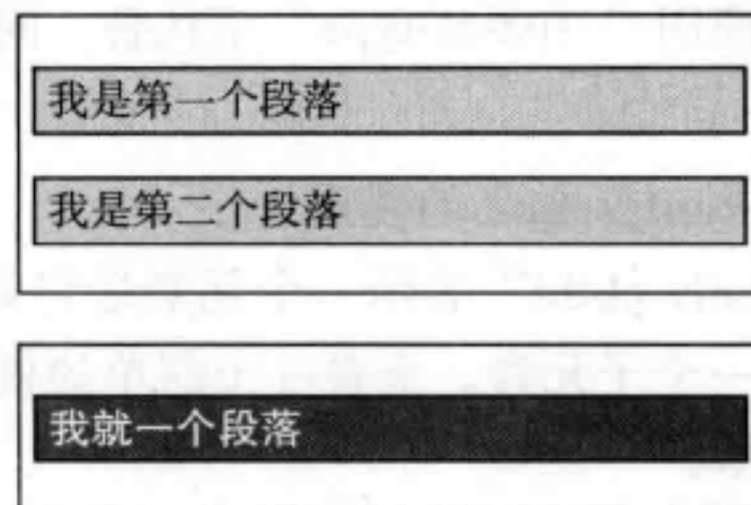


图 2-44 :only-child 器使用效果

很明显的两个列表中只有一个段落 p 的改变了样式，也就是说“:only-child”仅能匹配父元素中一个子元素，而且这个子元素是唯一的。

9. :only-of-type 的使用

“:only-of-type”用来选择一个元素是它的父元素的唯一一个相同类型的子元素。这样说或许不太好理解。换一种说法，“:only-of-type”表示一个元素有很多个子元素，而其中只有一个子元素是唯一的，使用“:only-of-type”就可以选中这个唯一类型子元素。

Devsnipet[⊖]制作了一个 demo，只有一张图片与一个容器中有多张图片的不同样式风格，下面代码是从 demo 中截取的。

```

div > img {
  background:#DCE8F5 none repeat scroll 0 0;
  border:3px solid #96C8E5;
  float:left;
}

```

⊖ 链接为 <http://devsnippets.com/article/5-advanced-css-pseudo-class.html>。


```

margin:5px;
padding:5px;
}
div > img:only-of-type {
border:3px solid #E71F58;
float:none;
margin:10px;
padding:10px;
}

```

从代码中可以明显的得知，div 中的图片左浮动，但 div 中仅有一图片类型时，此图片样式不浮动，而且还将改变对应的外边距和内边距值，效果如图 2-45 所示。



图 2-45 Devsnippet 实例效果

10. :empty 的使用

“:empty”用来选择没有任何内容的元素，这里“没有任何内容”指的是一点内容都没有，哪怕是一个空格。这个选择器用来处理动态输出内容方面非常方便。例如想高亮提示用户搜索出来的结果为空时，就可以这样使用。

```
#results:empty{background-color:#fcc;}
```

2.8.6 实战体验：CSS3 美化表格

对于数量大的表格，普通的设计极易影响用户的阅读体验。例如长时间地阅读数据量大的表格容易引起视觉的疲劳，会诱发错行误读等问题。因此，Web 设计师要设计一个表格，不仅需要考虑表格的外观，而且要提高用户的体验。

Zebra 是经典的数据表格设计样式，主要从易用性的角度来考虑，以提高用户浏览数据的速度和准确度。传统的做法是在表格的行中分奇数和偶数，为相应的行添加类名，然后通过类名来控制表格单元格的背景色。当然，这种设计给前端工作人员带来很多不便之处，例如动态插入行，就需要重新为行设置类名，也给维护带来很多困难。

然而采用 CSS3 结构伪类选择器后，一切都是那么简单而轻松。

```

<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>CSS3 美化表格 </title>
  <style type="text/css">
    *{margin: 0;padding: 0;}
  body {
    padding: 40px 100px;
  }
  .demo {

```

```

width: 600px;
margin: 40px auto;
font-family: 'trebuchet MS', 'Lucida sans', Arial;
font-size: 14px;
color: #444;
}
/* 表格的默认设置 */
table {
  *border-collapse: collapse; /* IE 7 and lower */
  border-spacing: 0;
  width: 100%;
}

/*===== 制作圆角表格 =====*/
.bordered {
  border: solid #ccc 1px; /* 给表格添加边框 */
  border-radius: 6px; /* 设置表格圆角 */
  box-shadow: 0 1px 1px #ccc; /* 表格阴影设置 */
}

.bordered tr {
  -o-transition: all 0.1s ease-in-out;
  -webkit-transition: all 0.1s ease-in-out;
  -moz-transition: all 0.1s ease-in-out;
  -ms-transition: all 0.1s ease-in-out;
  transition: all 0.1s ease-in-out;
}

.bordered .highlight,
.bordered tr:hover {
  background: #fbf8e9; /* 表格行的悬浮状态效果 */
}

.bordered td,
.bordered th {
  border-left: 1px solid #ccc;
  border-top: 1px solid #ccc;
  padding: 10px;
  text-align: left;
}

.bordered th {
  /* 表格表头添加渐变背景色 */
  background-color: #dce9f9;
  background-image: -webkit-gradient
    (linear, left top, left bottom, from(#ebf3fc), to(#dce9f9));
  background-image: -webkit-linear-gradient(top, #ebf3fc, #dce9f9);
  background-image: -moz-linear-gradient(top, #ebf3fc, #dce9f9);
  background-image: -ms-linear-gradient(top, #ebf3fc, #dce9f9);
  background-image: -o-linear-gradient(top, #ebf3fc, #dce9f9);
  background-image: linear-gradient(top, #ebf3fc, #dce9f9);
  filter: progid:DXImageTransform.Microsoft.gradient(GradientType=0,
    startColorstr=#ebf3fc, endColorstr=#dce9f9);
  -ms-filter: "progid:DXImageTransform.Microsoft.gradient (GradientType=0,
    startColorstr=#ebf3fc, endColorstr=#dce9f9)";
}

```



```

    box-shadow: 0 1px 0 rgba(255,255,255,.8) inset; /* 表格表头设置内阴影 */
    border-top: none;
    text-shadow: 0 1px 0 rgba(255,255,255,.5); /* 表格表头设置文字阴影 */
}
/* 使用 :first-child 去除表格每行的第一个单元格的左边框 */
.bordered td:first-child,
.bordered th:first-child {
    border-left: none;
}
/* 使用 :first-child 设置表格表头第一个单元格仅左上角为圆角 */
.bordered th:first-child {
    border-radius: 6px 0 0 0;
}
/* 使用 :last-child 设置表格表头最后一个单元格仅右上角为圆角 */
.bordered th:last-child {
    border-radius: 0 6px 0 0;
}
/* 使用 :first-child 和 :last-child 设置表格最后一行的第一个单元格左下角为圆角 */
.bordered tr:last-child td:first-child {
    border-radius: 0 0 0 6px;
}
/* 使用 :last-child 设置表格最后一行的最后一个单元格右上角为圆角 */
.bordered tr:last-child td:last-child {
    border-radius: 0 0 6px 0;
}

/*===== 制作 Zebra 表格 (斑马线表格) 效果 =====*/
.zebra td,
.zebra th {
    padding: 10px;
    border-bottom: 1px solid #f2f2f2;
}
/* 使用 :nth-child(even) 给表格的奇数行添加背景和阴影效果 */
.zebra .alternate,
.zebra tbody tr:nth-child(even) {
    background: #f5f5f5;
    box-shadow: 0 1px 0 rgba(255,255,255,.8) inset;
}
.zebra th {
    text-align: left;
    text-shadow: 0 1px 0 rgba(255,255,255,.5);
    border-bottom: 1px solid #ccc;
    background-color: #eee;
    background-image: -webkit-gradient(linear,
        left top, left bottom, from(#f5f5f5), to(#eee));
    background-image: -webkit-linear-gradient(top, #f5f5f5, #eee);
    background-image: -moz-linear-gradient(top, #f5f5f5, #eee);
    background-image: -ms-linear-gradient(top, #f5f5f5, #eee);
    background-image: -o-linear-gradient(top, #f5f5f5, #eee);
    background-image: linear-gradient(top, #f5f5f5, #eee);
    filter: progid:DXImageTransform.Microsoft.gradient(GradientType=0,

```

```

        startColorstr=#f5f5f5, endColorstr=#eeeeee);
    -ms-filter: "progid:DXImageTransform.Microsoft.gradient (GradientType=0,
        startColorstr=#f5f5f5, endColorstr=#eeeeee)";
}
/* 使用 :first-child 设置表格表头第一个单元格左上角为圆角 */
.zebra th:first-child {
    border-radius: 6px 0 0 0;
}
/* 使用 :last-child 设置表格表头最后一个单元格右上角为圆角 */
.zebra th:last-child {
    border-radius: 0 6px 0 0;
}
.zebra tfoot td {
    border-bottom: 0;
    border-top: 1px solid #fff;
    background-color: #f1f1f1;
}
/* 使用 :first-child 设置表格脚部第一个单元格左下角为圆角 */
.zebra tfoot td:first-child {
    border-radius: 0 0 0 6px;
}
/* 使用 :last-child 设置表格脚部最后一个单元格右下角为圆角 */
.zebra tfoot td:last-child {
    border-radius: 0 0 6px 0;
}
</style>
</head>
<body>
...
</body>
</html>

```

通过上面的代码，在现代浏览器中就能看到两个美化后的表格，如图 2-46 所示。

#	IMDB Top 10 Movies	Year
1	The Shawshank Redemption	1994
2	The Godfather	1972
3	The Godfather: Part II	1974
4	The Good, the Bad and the Ugly	1966

#	IMDB Top 10 Movies	Year
1	The Shawshank Redemption	1994
2	The Godfather	1972
3	The Godfather: Part II	1974
4	The Good, the Bad and the Ugly	1966

图 2-46 CSS3 美化表格

图 2-46 的表格效果是不是很清晰。我们需要知道这样的表格是怎么制作出来的, 下面就一起来细化以上代码。

以前表格的圆角效果是通过图片来模拟, 制作相当麻烦。有了 CSS3 后, 这些都变得那么容易, 只要使用 `border-radius` 就可以 (这个属性后面章节会详细介绍)。制作表格圆角时还有一个技巧, 在制作表格圆角效果之前, 有必要先完成一步。`border-collapse` 的默认值是 `separate`, 需要将其设置为 0, 如下所示。

```
table{
    *border-collapse: collapse; /*IE 7 and lower*/
    border-spacing: 0;
}
```

接下来一起看表格圆角实现的代码。

```
/*== 整个表格设置了边框, 并设置了圆角 ==*/
.bordered {
    border: solid #ccc 1px;
    border-radius: 6px;
}
/*== 表格头部第一个 th 需要设置一个左上角圆角 ==*/
.bordered th:first-child {
    border-radius: 6px 0 0 0;
}
/*== 表格头部最后一个 th 需要设置一个右上角圆角 ==*/
.bordered th:last-child {
    border-radius: 0 6px 0 0;
}
/*== 表格最后一行的第一个 td 需要设置一个左下角圆角 ==*/
.bordered tr:last-child td:first-child {
    border-radius: 0 0 0 6px;
}
/*== 表格最后一行的最后一个 td 需要设置一个右下角圆角 ==*/
.bordered tr:last-child td:last-child {
    border-radius: 0 0 6px 0;
}
```

在 `table` 中设置一个边框, 为了让表格具有圆角效果, 需要在表格四个角的单元格上分别设置圆角效果, 并且其圆角的半径弧度与表格的圆角半径弧度大小一样。反之, 如果在 `table` 上没有设置边框, 只需要在表格四个角的单元格设置圆角, 就能实现圆角效果, 例如 Zebra 表格。

```
/*== 表格头部第一个 th 需要设置一个左上角圆角 ==*/
.zebra th:first-child {
    border-radius: 6px 0 0 0;
}
/*== 表格头部最后一个 th 需要设置一个右上角圆角 ==*/
.zebra th:last-child {
```

```
border-radius: 0 6px 0 0;
}
/*== 表格最后一行的第一个 td 需要设置一个左下角圆角 ==*/
.zebra tfoot td:first-child {
border-radius: 0 0 0 6px;
}
/*== 表格最后一行的最后一个 td 需要设置一个右下角圆角 ==*/
.zebra tfoot td:last-child {
border-radius: 0 0 6px 0;
}
```

例中的表格除了使用了 CSS3 的圆角效果之外，还使用了 box-shadow 制作表格的阴影效果，使用 text-shadow 制作表格表头的文字阴影效果，使用 transition 制作 hover 悬浮高亮的过渡效果，以及使用 gradient 制作表头的渐变效果。这些属性还不清楚如何使用，大家不必太担心，本书后续章节会一一介绍。

2.9 否定伪类选择器

否定选择器 “:not()” 是 CSS3 的新选择器，类似 jQuery 中的 “:not()” 选择器，主要用来定位不匹配该选择器的元素。

2.9.1 否定伪类选择器语法

“:not()” 是一个非常有用的选择器，可以起到过滤内容的作用。语法如表 2-16 所示。

表 2-16 否定伪类选择器的使用语法	
选择器	功能描述
E:not(F)	匹配所有除元素 F 外的 E 元素

否定选择器作用非常大，例如以下选择器表示选择页面中所有元素，除了 “footer” 元素之外。

```
:not(footer){...}
```

有时候常在表单元素中使用，举个实例，给表单中所有 input 定义样式，除了 submit 按钮之外，此时就可以使用否定选择器。

```
input:not([type=submit]){...}
```

类似这样的选择器在移动端使用也是常见的，例如在 Web 移动页面中，给表单中的 input 定义样式，除了单选择按钮之外，代码如下所示。

```
fieldset input:not([type=radio] {
margin:0;
width:265px;
font-size: 18px;
border-radius: 0;
border-bottom: 0;
```








```
border-color: #999;
padding: 8px 10px;
}
```

2.9.2 浏览器兼容性

浏览器兼容性如表 2-17 所示。

表 2-17 否定伪类选择器的兼容性

选择器					
E:not(F)	9 + √	√	√	√	√

2.9.3 实战体验：改变图片效果

本节的实例是通过否定选择器来改变图片墙中图片，用来区分悬浮状态下的效果。
这个实例中采用两种技术，一种是图片的过滤效果（CSS3 中的新特性，现在仅 Webkit 内核浏览器支持，本例子中不详细介绍），第二种技术就是前面介绍的否定选择器“:not”。
当鼠标悬浮在整个图片墙上时，所有图片通过自定义的过滤特性，变成黑白模糊的效果，当鼠标移动到一张图片上时，图片恢复到默认效果，而其他图片保持黑白模糊效果，如图 2-47 所示。

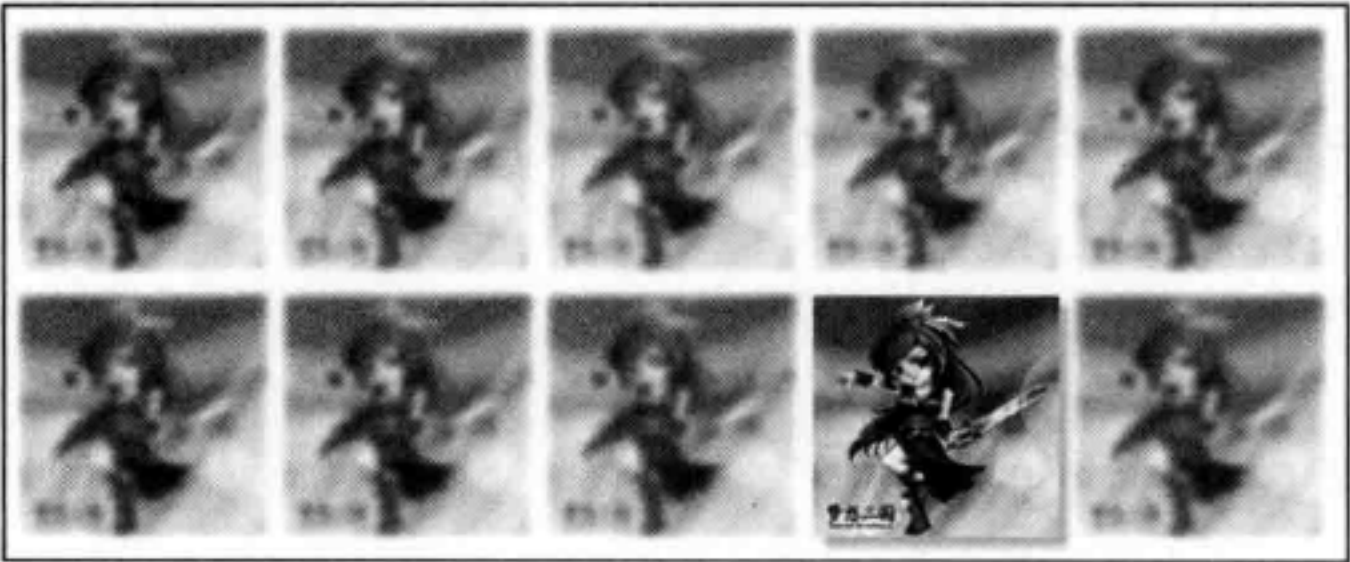


图 2-47 否定选择器制作图片墙

制作原理非常简单，接下来一起来看看如何实现。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>:not 选择器使用 </title>
  <style type="text/css">
    *{
      margin: 0;
      padding: 0;
    }
    ul {
      width: 690px;
      margin: 20px auto;
      letter-spacing: -4px;
      word-spacing: -4px;
    }
  </style>
</head>
```

```

    font-size: 0;
  }
  li {
    font-size: 16px;
    letter-spacing: normal;
    word-spacing: normal;
    display: inline-block;
    *display: inline;
    zoom: 1;
    list-style: none outside none;
    margin: 5px;
  }
  img {
    width: 128px;
    height: 128px;
  }
  .gallery li:nth-child(2){
    -webkit-filter: grayscale(1);
  }
  .gallery li:nth-child(3){
    -webkit-filter: sepia(1);
  }
  .gallery li:nth-child(4){
    -webkit-filter: saturate(0.5);
  }
  .gallery li:nth-child(5){
    -webkit-filter: hue-rotate(90deg);
  }
  .gallery li:nth-child(6){
    -webkit-filter: invert(1);
  }
  .gallery li:nth-child(7){
    -webkit-filter: opacity(0.2);
  }
  .gallery li:nth-child(8){
    -webkit-filter: blur(3px);
  }
  .gallery li:nth-child(9){
    -webkit-filter: drop-shadow(5px 5px 5px #ccc);
  }
  .gallery li:nth-child(10){
    -webkit-filter: saturate(6) hue-rotate(361deg) brightness(.09);
  }
  .gallery: hover li: not(:hover){
    -webkit-filter: blur(2px) grayscale(1);
    opacity: .7;
  }
</style>
</head>
<body>
  <ul class="gallery">

```



```

<li>
  
</li>
...
<li>
  
</li>
</ul>
</body>
</html>

```

整个案例中，通过“:nth-child()”给每个图片设置 filter 效果，关键部分是使用“:not()”过滤了除悬浮状态“(:hover)”的图片，其他都变成黑白模糊透明度为 70% 的效果。

```

.gallery:hover li:not(:hover){
  -webkit-filter: blur(2px) grayscale(1);
  opacity: .7;
}

```

不过在写本书时，仅有 Webkit 内核的浏览器支持 filter 属性。当浏览器不支持 filter，但支持“:not()”，能看到除悬浮状态图片以外的所有图片透明度变成 70%；如果对否定选择器也不支持，将看不到任何的效果。

2.10 伪元素

除了伪类，CSS3 还支持访问伪元素。伪元素可用于定位文档中包含的文本，但无法在文档树中定位。伪类一般反映无法在 CSS 中轻松或可靠地检测到的某个元素属性或状态；另一方面，伪元素表示 DOM 外部的某种文档结构。

伪元素其实在 CSS 中一直存在，大家平时看到的有“:first-line”、“:first-letter”、“:before”和“:after”。CSS3 中对伪元素进行了一定的调整，在以前的基础上增加一个冒号，也就相应的变成了“::first-letter”、“::first-line”、“::before”和“::after”，另外伪元素还增加了一个“::selection”。

或许大家会问，为什么要使用两个冒号？对于 IE 6 ~ 8，仅支持单冒号表示法，而现代浏览器同时支持这两种表示法。另外一个区别是，双冒号与单冒号在 CSS3 中主要用来区分伪类和伪元素。到目前来说，这两种方式都是被浏览器接受的。接下来简单介绍一下伪类元素的作用。

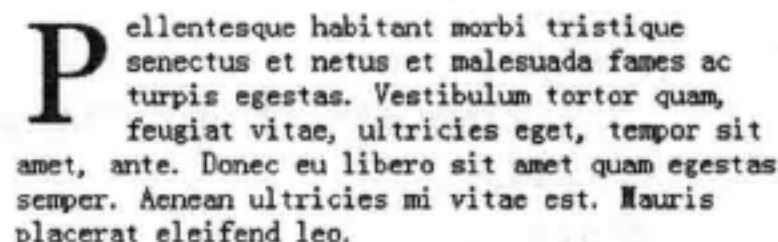
2.10.1 伪元素 ::first-letter

“::first-letter”用来选择文本块的第一个字母，除非在同一行中包含一些其他元素。“::first-letter”通常用于给文本元素添加排版细节，例如下沉字母或首字母，下面的代码是

如何使用“`::first-letter`”创建首字下沉。

```
p:first-child::first-letter {
  float: left;
  color: #903;
  padding: 4px 8px 0 3px;
  font: 75px/60px Georgia;
}
```

效果如图 2-48 所示。



Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo.

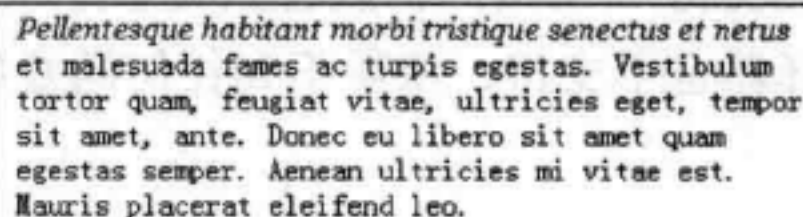
图 2-48 `::first-letter` 制作首字下沉效果

2.10.2 伪元素 `::first-line`

“`::first-line`”的使用和“`::first-letter`”类似，也常用于文本排版方面，只不过“`::first-line`”用来匹配元素的第一行文本，可以应用一些特殊的样式，给文本添加一些细微的区别。“`::first-line`”将匹配 block、inline-block、table-caption、table-cell 等级别元素的第一行，来看一个简单的例子。

```
p:last-child::first-line {
  font: italic 16px/18px Georgia;
  color: #903;
}
```

上面的代码意味着最后一个段落的第一行文字显示为红色斜体，如图所示 2-49 所示。



Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo.

图 2-49 `::first-line` 制作首行文字效果

2.10.3 伪元素 `::before` 和 `::after`

对于“`::before`”和“`::after`”来说，大家并不多见，但“`:before`”和“`:after`”，或许不会陌生，因为清除浮动就使用这两个伪类。

“`::before`”和“`::after`”不是指存在于标记中的内容，而是可以插入额外内容的位置。尽管生成的内容不会成为 DOM 的一部分，但它同样可以设置样式。

要为伪元素生成内容，还需要配合 content 属性。例如，假设在页面上所有外部链接之后的括号中附加它们所指向的 URL，无须将 URL 硬编码到标记中，可以结合使用一个属性选择器和“`::after`”伪元素。

```
a[href^=http]::after {
  content: "(" attr(href) " " ";
}
```

如今在 CSS3 中使用“`::before`”和“`::after`”越来越多见，例如给链接添加 ICON 的效果。Font Awesome^① 站点制作的 ICON 就使用伪元素“`::before`”和“`::after`”，下面截取部

① 网址为 <http://fontawesome.github.com/Font-Awesome>。

分代码。

```

/* Font Awesome styles
----- */
[class^="icon-"]:before,
[class*=" icon-"]:before {
    font-family: FontAwesome;
    font-weight: normal;
    font-style: normal;
    display: inline-block;
    text-decoration: inherit;
}
a [class^="icon-"],
a [class*=" icon-"] {
    display: inline-block;
    text-decoration: inherit;
}
/* makes the font 33% larger relative to the icon container */
.icon-large:before {
    vertical-align: middle;
    font-size: 1.3333333333333333em;
}
.btn [class^="icon-"],
.nav-tabs [class^="icon-"],
.btn [class*=" icon-"],
.nav-tabs [class*=" icon-"] {
    /* keeps button heights with and without icons the same */

    line-height: .9em;
}
li [class^="icon-"],
li [class*=" icon-"] {
    display: inline-block;
    width: 1.25em;
    text-align: center;
}
li .icon-large:before,
li .icon-large:before {
    /* 1.5 increased font size for icon-large * 1.25 width */

    width: 1.875em;
}
ul.icons {
    list-style-type: none;
    margin-left: 2em;
    text-indent: -0.8em;
}
ul.icons li [class^="icon-"],
ul.icons li [class*=" icon-"] {
    width: .8em;
}

```

```

ul.icons li .icon-large:before,
ul.icons li .icon-large:before {
  /* 1.5 increased font size for icon-large * 1.25 width */
  vertical-align: initial;
}
.icon-bullhorn::before {
  content: "\f0a1";
}
.icon-beaker::before {
  content: "\f0c3";
}

```

效果如图 2-50 所示。



图 2-50 Font Awesome 制作 ICON



注意

图 2-50 效果还需要配合 CSS3 的 @font-face 特性，后续章节会进行详细介绍。

2.10.4 伪元素 ::selection

“::selection”是用来匹配突出显示的文本。浏览器默认情况下，选择网站文本是深蓝的背景，白色的字体，如图 2-51 所示。

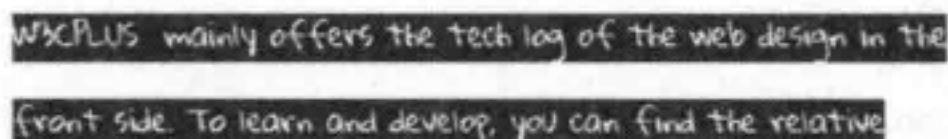


图 2-51 浏览器默认突出文本的效果

有的设计需要一个与众不同的效果，此时“::selection”就非常实用。不过浏览器对“::selection”支持并不完美，在整个 IE 系列中仅有 IE 9 支持，Firefox 也需要加上其私有属性“-moz”。不过值得庆幸的是，Webkit 内核浏览器支持，其正确的使用如下。

```

/*Webkit,Opera9.5+,IE 9+*/
::selection {
  background: 颜色值;
  color: 颜色值;
}

```



```
}
/*Mozilla Firefox*/
::-moz-selection {
    background: 颜色值;
    color: 颜色值;
}
```

一起来看看 W3cplus¹² 上的使用。

```
:::selection {
    background: red;
    color: #fff;
}
::-moz-selection {
    background: red;
    color: #fff;
}
```

此时选择文本时，背景是大红色，前景色是白色，如图 2-52 所示。

基于JQuery实现的当当品牌选择器的功能的实现功能描述：1、默认状态下品牌只显示其中的一部分，但是实事上所有的品牌的名称和逻辑

图 2-52 ::selection 伪元素突出文本效果



注意 伪元素 ::selection 仅接受两个属性，一个是 background，另一个是 color。

2.11 属性选择器

在 HTML 中，通过各种各样的属性可以给元素增加很多附加的信息。例如，通过 id 属性可以将不同 DIV 元素进行区分。CSS2 中引入了一些属性选择器，这些选择器可基于元素的属性来匹配元素，而 CSS3 在 CSS2 的基础上扩展了这些属性选择器，支持基于模式匹配来定位元素。

2.11.1 属性选择器语法

CSS3 在 CSS2 的基础上新增了 3 个属性选择器，可以帮助大家对标签进行过滤，也能非常容易帮助大家在众多标签中定位自己需要的 HTML 标签，下面通过表 2-18 详细介绍 CSS3 的属性选择器的使用。

表 2-18 CSS3 属性选择器列表

选择器	功能描述
E[attr]	选择匹配具有属性 attr 的 E 元素。其中 E 可以省略，表示选择定义了 attr 属性的任意类型元素

(续)

选择器	功能描述
E[attr=val]	选择匹配具有属性 attr 的 E 元素，并且 attr 的属性值为 val（其中 val 区分大小写），同样 E 元素省略时表示选择定义了 attr 属性值为 val 的任意类型元素
E[attr =val]	选择匹配 E 元素，且 E 元素定义了属性 attr，attr 属性值是一个具有 val 或者以 val- 开始的属性值。常用于 lang 属性（例如 lang=" en-us"）。例如 p[lang=en] 将匹配定义为英语的任何段落，无论是英式英语还是美式英语
E[attr~=val]	选择匹配 E 元素，且 E 元素定义了属性 attr，attr 属性值具有多个空格分隔的值，其中一个值等于 val。例如，.info[title~more] 将匹配元素具有类名 info，而且这个元素设置了一个属性 title，同时 title 属性值以包含了“ more”的任何元素，例如 click me
E[attr*=val]	选择匹配元素 E，且 E 元素定义了属性 attr，其属性值任意位置包含了“ val”。换句话说，字符串 val 与属性值中的任意位置相匹配
E[attr^=val]	选择匹配元素 E，且 E 元素定义了属性 attr，其属性值以 val 开头的任何字符串
E[attr\$=val]	选择匹配元素 E，且 E 元素定义了属性 attr，其属性值以 val 结尾的任何字符串。刚好与 E[attr^=val] 相反

CSS3 遵循了惯用的编码规则，通配符的使用提高了样式表的书写效率，也使 CSS3 的属性选择器更符合编码习惯。CSS3 中常见的通配符如表 2-19 所示。






表 2-19 CSS3 中常见的通配符

通配符	功能描述	示例
^	匹配起始符	span[class^=span] 表示选择以类名以“span”开头的所有 span 元素
\$	匹配终止符	a[href\$=pdf] 表示选择以“ pdf”结尾的 href 属性的所有 a 元素
*	匹配任意字符	a[title*=site] 匹配 a 元素，而且 a 元素的 title 属性值中任意位置有“site”字符的任何字符串

2.11.2 浏览器兼容性

属性选择器在浏览器兼容性方面表现还是可以的，如表 2-20 所示。

表 2-20 属性选择器的浏览器兼容性

选择器					
E[attr]	7 + √	√	√	√	√
E[attr=val]	7 + √	√	√	√	√
E[attr =val]	7 + √	√	√	√	√
E[attr~=val]	7 + √	√	√	√	√
E[attr*=val]	7 + √	√	√	√	√
E[attr^=val]	7 + √	√	√	√	√
E[attr\$=val]	7 + √	√	√	√	√

2.11.3 属性选择器的使用方法详解

前面了解了 CSS3 属性选择器的使用规则以及浏览器兼容性，接下来通过一些简单的示例来体验一下 CSS3 属性选择器的强大功能。

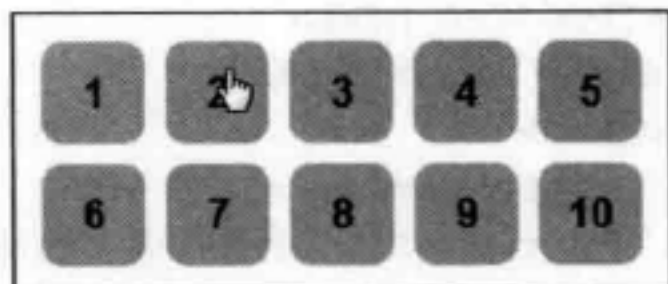
先创建一个简单的 HTML 结构，并付上一些默认样式。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>CSS3 属性选择器的使用 </title>
  <style type="text/css">
    .demo {
      width: 300px;
      border: 1px solid #ccc;
      padding: 10px;
      overflow: hidden;
      margin: 20px auto;
    }
    .demo a {
      float: left;
      display: block;
      height: 50px;
      width: 50px;
      border-radius: 10px;
      text-align: center;
      background: #aac;
      color: blue;
      font: bold 20px/50px Arial;
      margin-right: 5px;
      text-decoration: none;
      margin: 5px;
    }
  </style>
</head>
<body>
  <div class="demo">
    <a href="http://www.w3cplus.com" target="_blank"
      class="links item first" id="first" title="w3cplus">1</a>
    <a href="" class="links active item" title="test
      website" target="_blank" lang="zh">2</a>
    <a href="sites/file/test.html" class="links item"
      title="this is a link" lang="zh-cn">3</a>
    <a href="sites/file/test.png" class="links item"
      target="_blank" lang="zh-tw">4</a>
    <a href="sites/file/image.jpg" class="links item"
      title="zh-cn">5</a>
    <a href="mailto:w3cplus@hotmail" class="links item"
      title="website link" lang="zh">6</a>
```

```

<a href="/a.pdf" class="links item"
    title="open the website" lang="cn">7</a>
<a href="/abc.pdf" class="links item"
    title="close the website" lang="en-zh">8</a>
<a href="abcdef.doc" class="links item"
    title="http://www.sina.com">9</a>
<a href="abd.doc" class="linksitem last" id="last">10</a>
</div>
</body>
</html>

```



基本样式的效果如图 2-53 所示。

接下来使用 CSS3 属性选择器进行个性化修改。

图 2-53 CSS3 属性选择器使用默认效果

1. E[attr] 属性选择器

这个属性选择器是最简单的一种。用来选择有某个属性的元素，而不论这个属性值是什么。例如，选择所有带有 ID 属性的 a 元素，并且改变了匹配元素的背景色，如下所示。

```
a[id]{background-color:yellow;}
```

对照上面的 HTML 结构不难发现，只有第一个和最后一个链接元素 a 使用了 ID 属性，所以上面的代码选择了两个 a 元素，效果如图 2-54 所示。

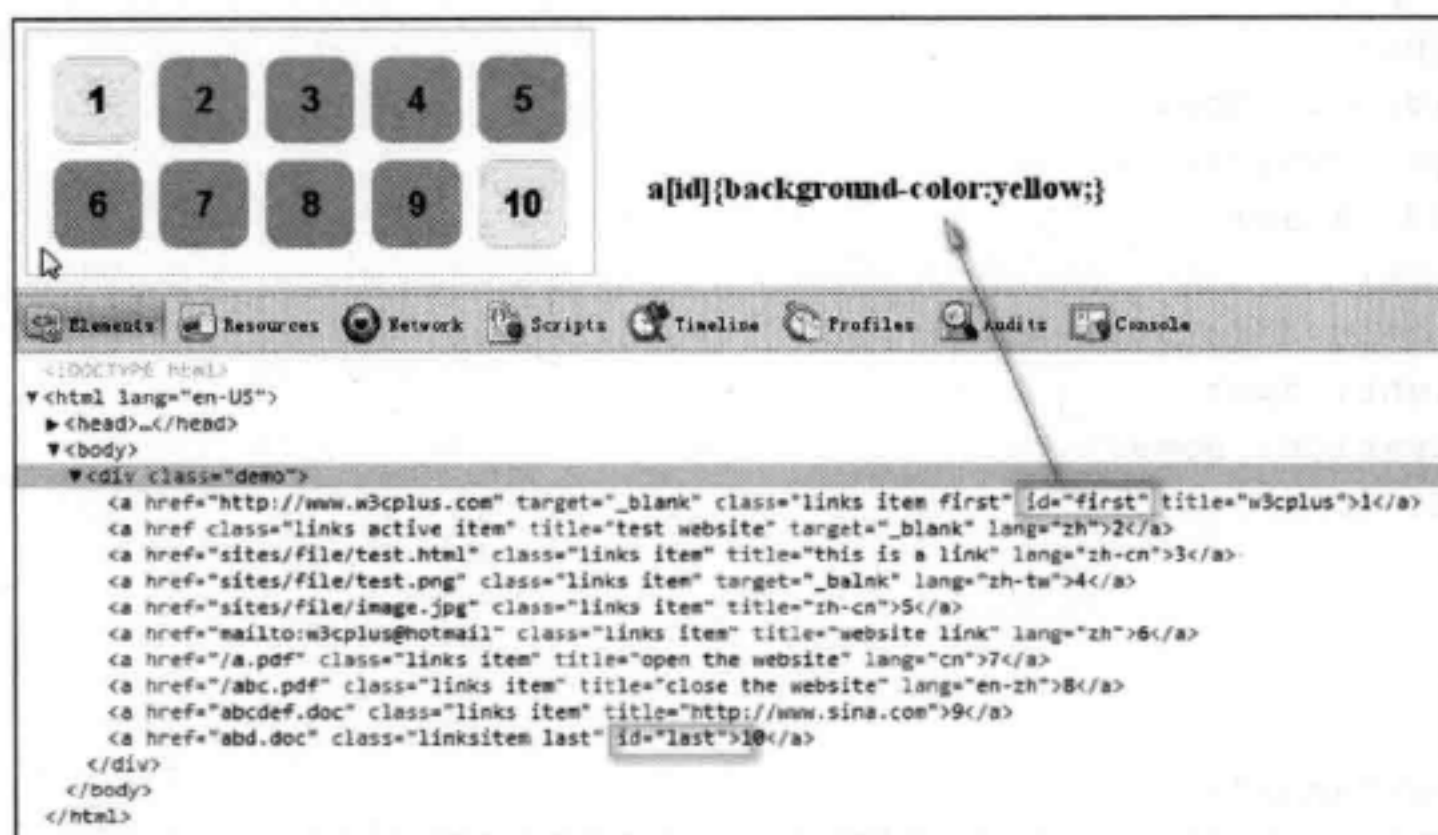


图 2-54 E[attr] 属性选择器的效果

也可以使用多属性进行选择元素。例如：

```
a[id][title]{background-color: red;}
```

上面代码表示元素 a 只要同时定义了 id 和 title 属性，都将匹配，其背景将变成红色，效果如图 2-55 所示。

2. E[attr=val] 属性选择器

E[attr=val] 是指元素 E 设置了属性 attr，并且其属性值为“val”，而 E[attr] 属性选择器

只选择了属性为 attr 的元素 E，并没有明确的设置 attr 的属性值，这也是这两种选择器的最大区别之处。在众多元素之中缩小选择范围，能进一步精确选择自己需要的元素。在前面的实例基础上进行一下简单的修改。

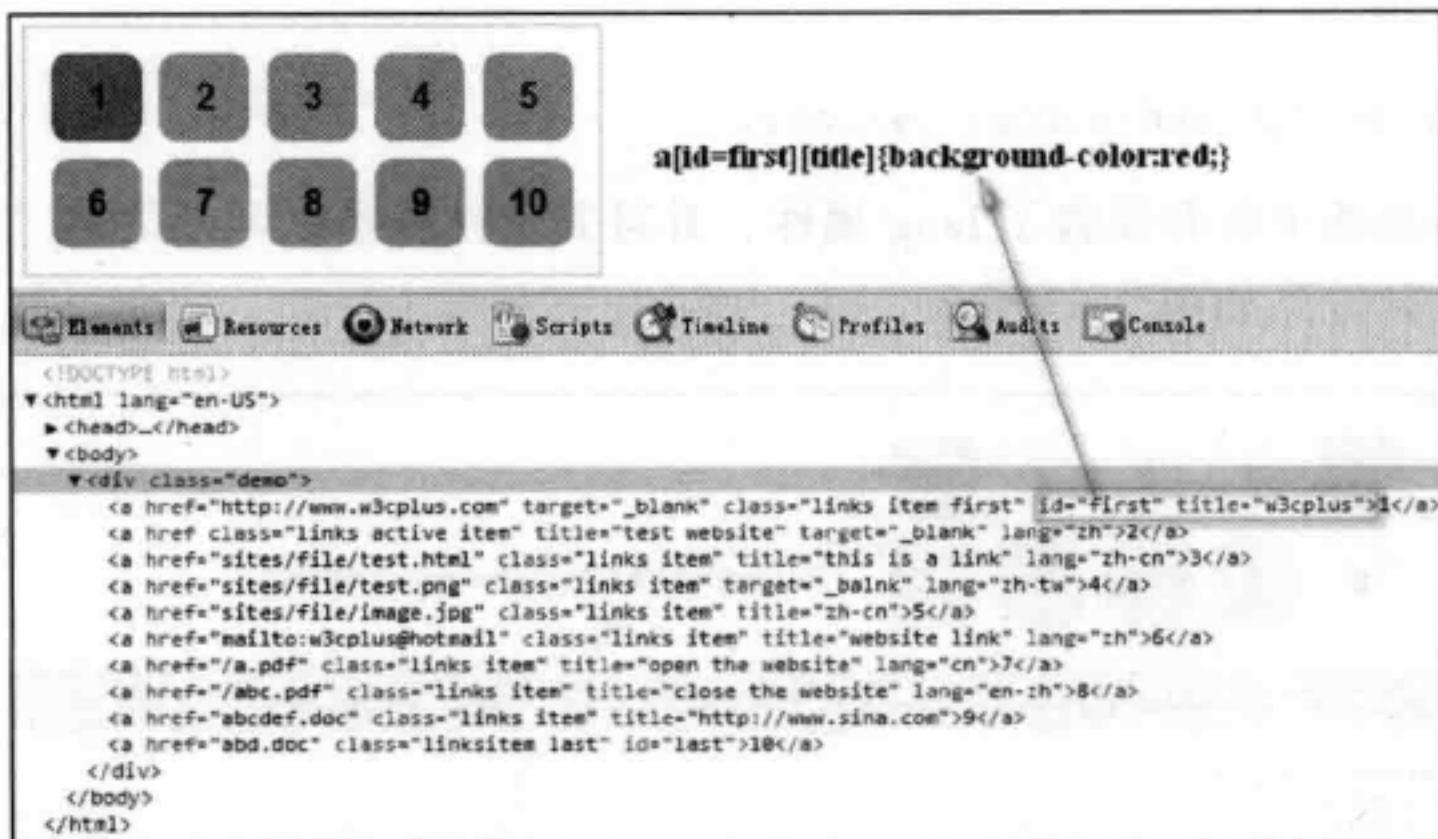


图 2-55 E[attr1][attr2] 多属性选择器使用效果

```
a[id=first]{background-color:red;}
```

此处是在 id 属性的基础上指定了相应的 val 值为 first，选中所有 a 元素中设置“id=first”的链接，并将其背景色设置为 red，效果如图 2-56 所示。

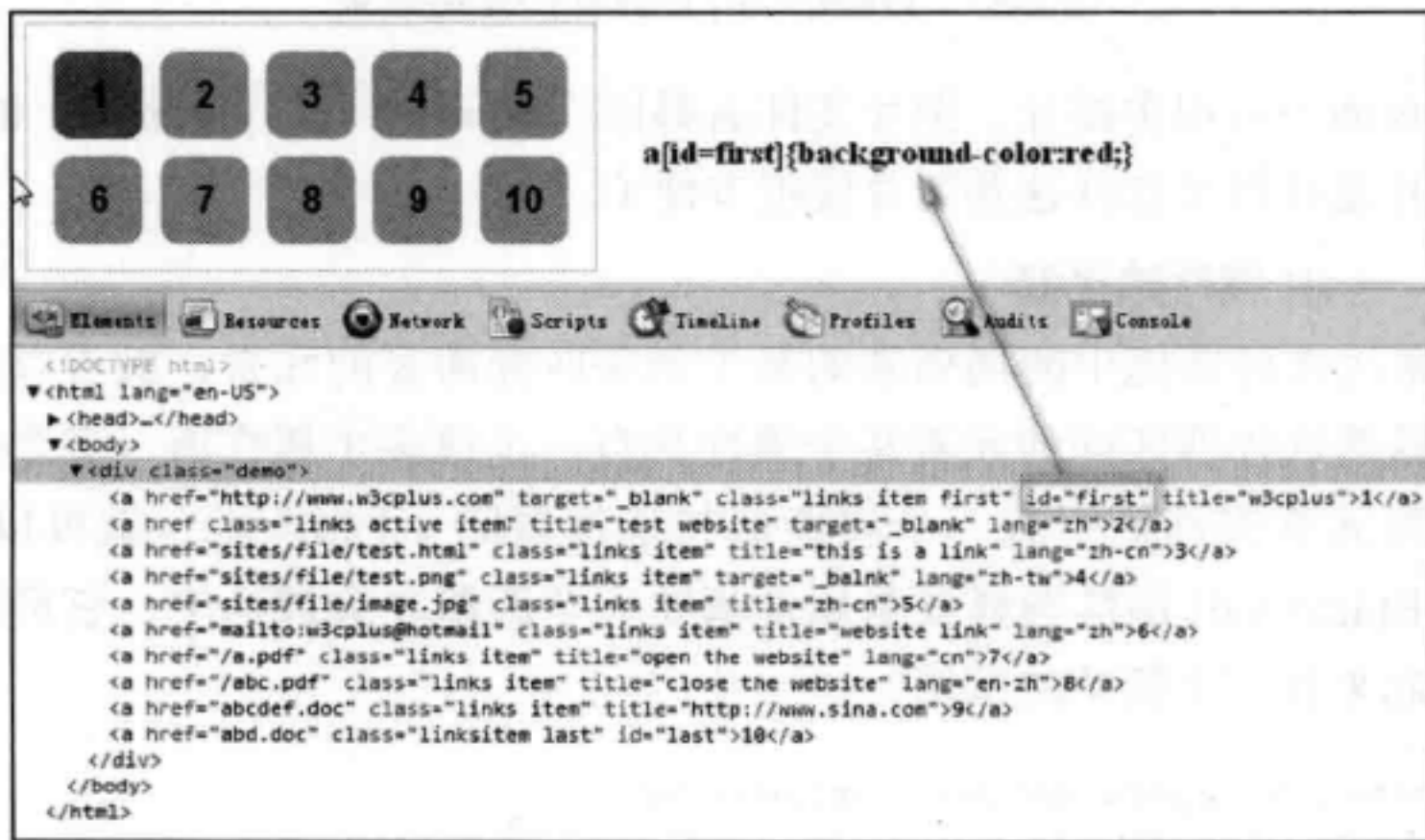


图 2-56 E[attr=val] 属性选择器的使用效果



注意 E[attr=val] 选择器中，属性和属性值必须完全匹配，特别对于属性值是词列表的形式，例如，

```
<a href="#" class="links item"></a>
```

其中 a[class="links"]{...} 是找不到匹配元素，只有 a[class="links item"]{...} 才匹配。

3. E[attr|=val] 属性选择器

首先提醒大家，attr 后面的是一条竖线，而不是字母。E[attr|=val] 为特定属性选择器，选择 attr 属性值等于“val”，或以“val-”开头的字符串属性的元素，一起来看看这个属性选择器的使用。

```
a[lang|=zh]{background-color: yellow;}
```

上面的代码会选中所有设置了 lang 属性，并且其属性值是以“zh”或“zh-”开头的所有 a 链接元素，其效果如图 2-57 所示。

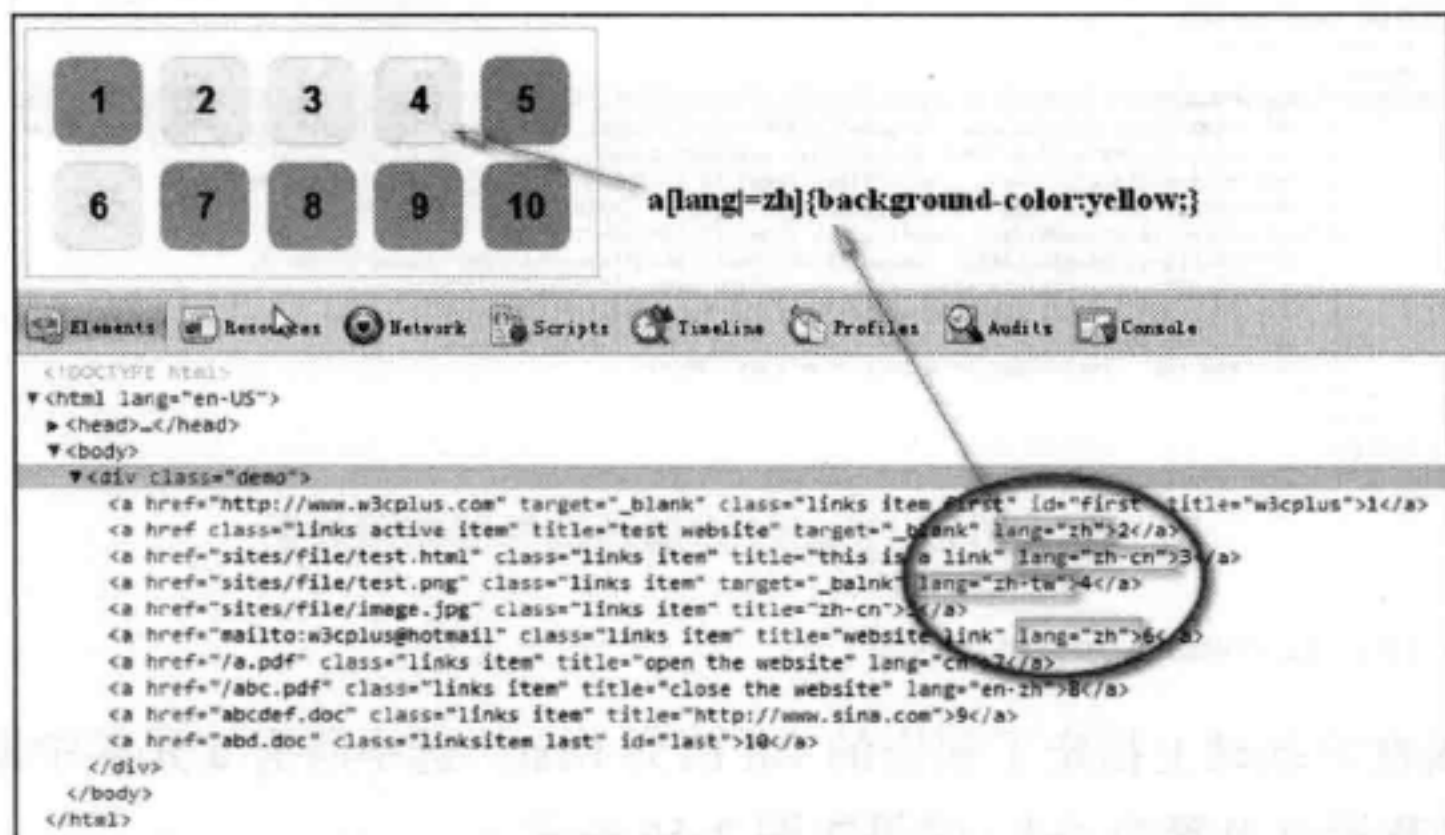


图 2-57 E[attr|=val] 选择器的使用效果

例如 Web 页面中有很多图片，图片文件名都以“figure-1”、“figure-2”的方式来命名，使用 E[attr|=val] 选择器来选择这些图片就很方便了。

4. E[attr~=val] 属性选择器

如果想根据元素属性值中的词列表的某个词来匹配需要的元素，就可以使用这个属性选择器。这种属性选择器匹配的元素某个属性具有一个或多个属性值，多个属性值之间使用空格隔开，当元素属性值中有一个属性值与选择器的 val 相匹配，就可以选中该元素。而前面介绍的 E[attr|=val] 属性选择器是属性值要完全匹配才会被选中，它们两者区别就是“~”符号。一起来看一个简单的示例。

```
a[title~=website]{background-color: yellow;}
```

效果如图 2-58 所示。

图 2-58 很清楚地告诉我们，只要 a 链接元素设置有 title 属性，并且这个 title 属性的值包含了 website 属性值，都将匹配。如果不小心将“~”省略，结果将找不到匹配的元素。

5. E[attr*=val] 属性选择器

这个属性选择器使用了通配符，将匹配元素设置了 attr 属性，而且 attr 属性值中包含

“val”字符串。也就是说，只要所选择的属性中有 val 字符串，都将被匹配。例如：

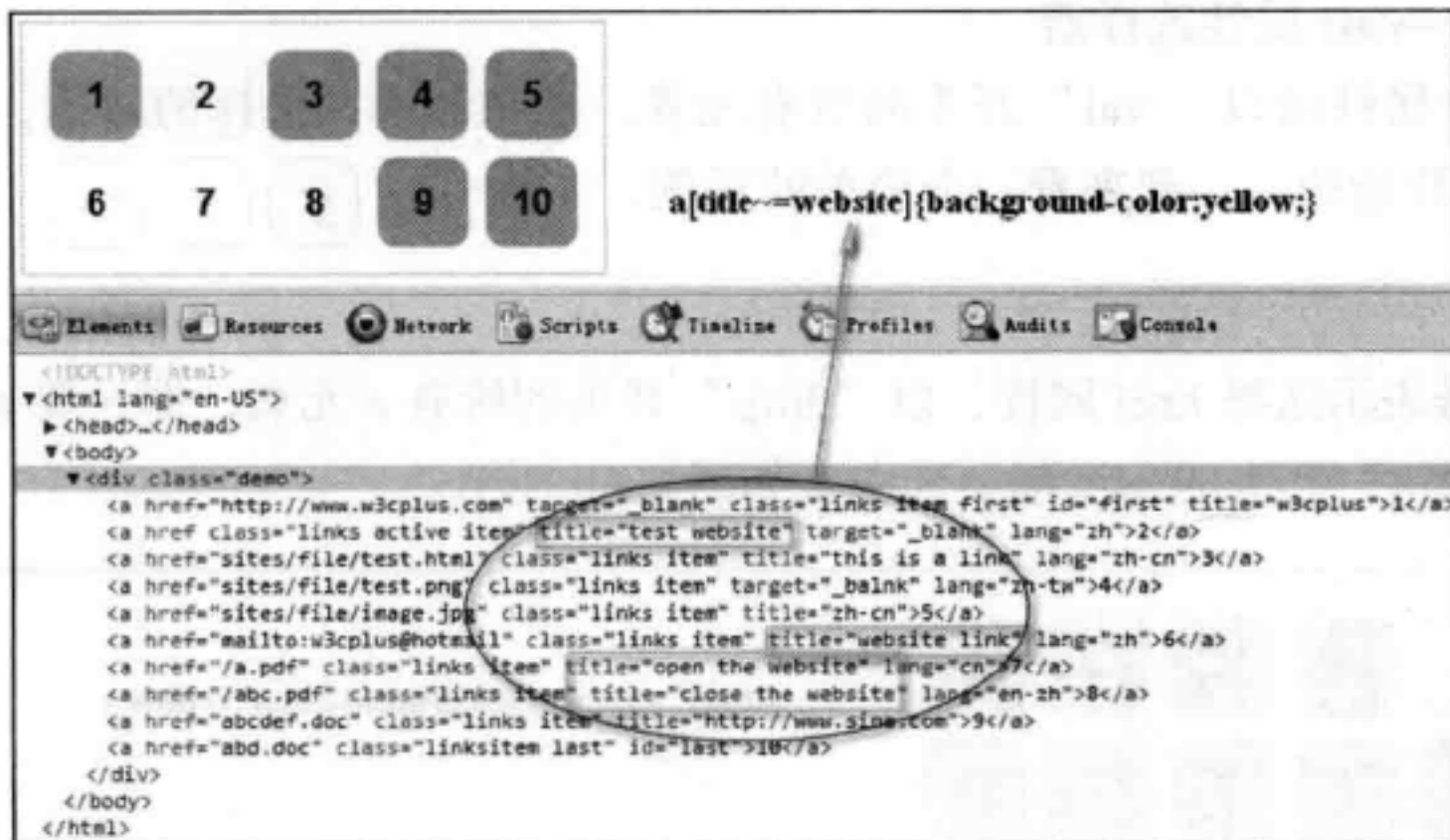


图 2-58 E[attr~=val] 选择器的使用效果

```
a[class*=links]{background-color:yellow;}
```

这个示例将选中所有的 a 元素，因为示例中的 a 元素“class”的属性值都含有字符串“links”，不管是“links”还是“linksitem”，其效果如图 2-59 所示。

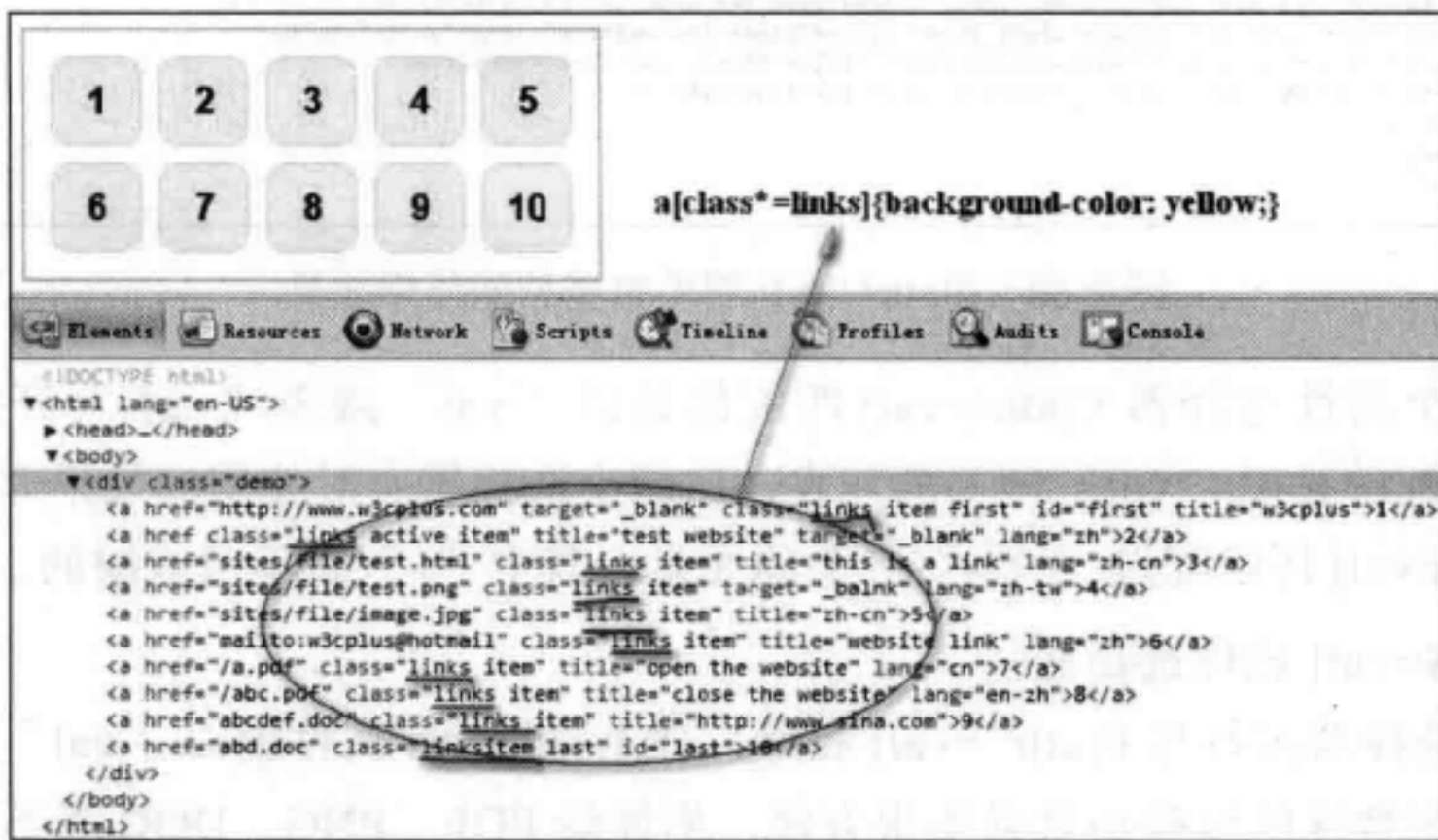


图 2-59 E[attr*=val] 属性选择器的使用效果

E[attr*=val] 与 E[attr~=val] 很容易混淆，它们的区别是：E[attr*=val] 匹配的是元素属性值中只要包含“val”字符串就可以，而 E[attr~=val] 匹配的是元素属性值中要包含“val”，并不仅是字符串。例如，a[title~=links] 属性值中的 links 是一个单词，而 a[title*=links] 中的 links 不一定是一个单词，就如上面的示例中，可以是“linksitem”。换句话说来说，“”元素只有 a[title*=links] 匹配，但是“<a title='links

item'>”元素，`a[title~=links]` 和 `a[title*=links]` 都匹配。

6. E[attr^=val] 属性选择器

指选择 attr 属性值以 “val” 开头的元素。换句话说，选择的属性，其对应的属性值是以 “val” 开始的，一起来看一个简单的示例。

```
a[href^=http]{background-color:yellow;}
```

这个选择器表示选择 href 属性，以 “http” 开头的元素，即只要 a 元素中的 href 属性值是以 “http” 开头的元素都会选中，如图 2-60 所示。

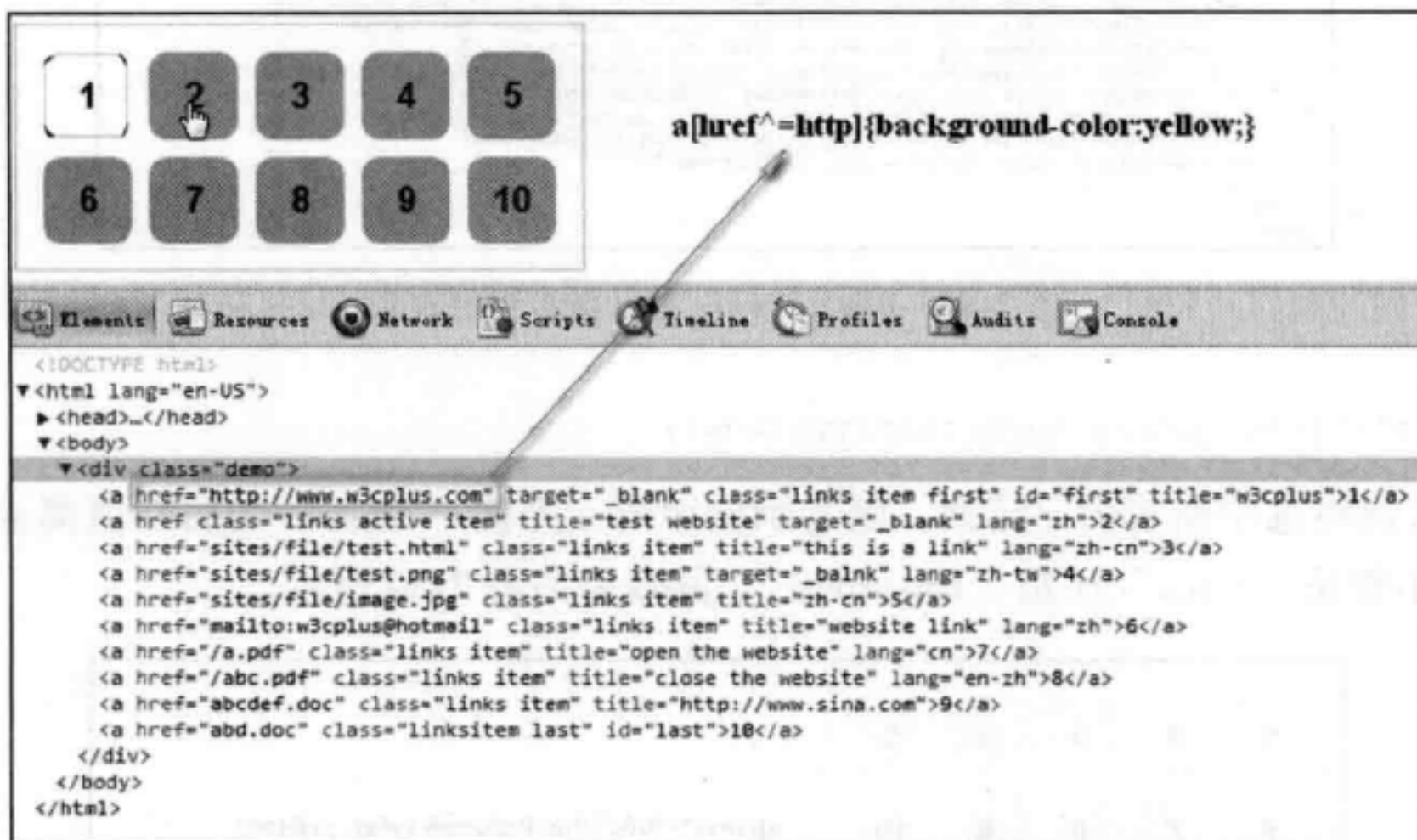


图 2-60 E[attr^=val] 属性选择器的使用效果

前面有一个属性选择器 `E[attr|=val]` 匹配的是以 “val” 或者 “val-” 开头的元素，而 `E[attr^=val]` 匹配的是以 “val” 开头的元素，这两个选择器有时使用的效果非常相似，仅仅区别在于 `E[attr|=val]` 还匹配以 “val-” 开头的元素，其中 “-” 是不可或缺的。

7. E[attr\$=val] 属性选择器

这个属性选择器刚好与 `E[attr^=val]` 相反，表示选择 attr 属性值以 “val” 结尾的所有元素。运用在一些特殊的链接加背景图很方便，例如给 PDF、PNG、DOC 等不同文件加上不同的 ICON 图标就可以使用这个属性。下面通过给 “png” 值结尾的 a 元素改变背景色。

```
a[href$=png]{background-color:yellow;}
```

运用上面代码的效果如图 2-61 所示。

通过一些简单的示例，向大家详细介绍了 CSS3 属性选择器的使用方法，以及效果，接下来通过实战来学习 CSS3 属性选择器在 Web 中是如何发挥作用。

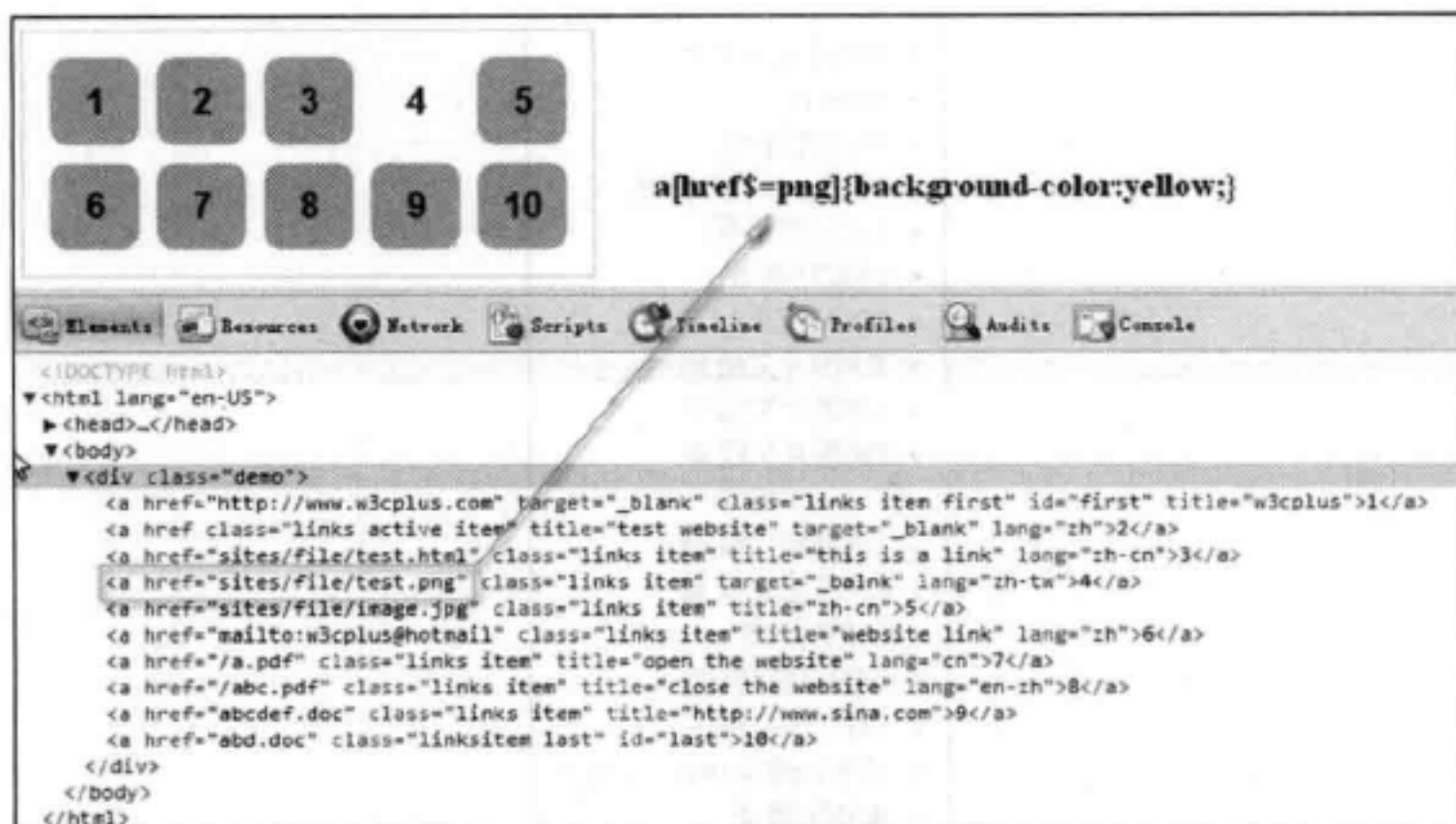


图 2-61 E[attr\$=val] 属性选择器的使用效果

2.11.4 实战体验：创建个性化链接样式

对于 Web 设计人员来说碰到的链接有很多种，例如文字、图片等。

例如新浪爱问共享资料库^①，下载文档列表的链接前面会显示一个文档类型图标（如图 2-62 所示），这样的设计给用户带来一种超好的体验。另外，有些 Web 网站对于本地链接和外部链接也会使用不同的图标来区分（如图 2-63 所示）。CSS3 的属性选择器此时就显得特别的有用，通过它们可以很容易实现这些效果。



图 2-62 显示类型图标的文档下载链接

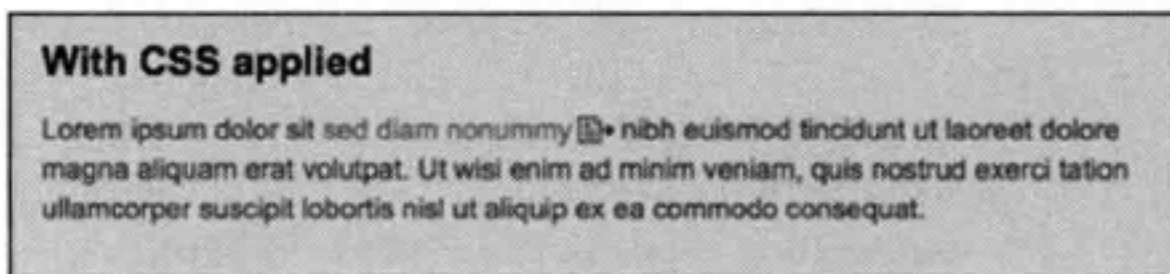


图 2-63 Web 页面的外部链接

本节示例的效果如图 2-64 所示，当然此效果只是部分 CSS3 属性选择器的运用。

图 2-62 和图 2-63 的链接效果，使用属性选择器匹配 a 元素中的 href 属性值。如果 a 元素的 href 属性值是以“http://”开头，将这些链接划分到外部链接一类中，给其定义一个样式，对于下载文档的类型可以根据其文件的扩展名来设置不同的图标。例如，链接为 Word 文件时，后面显示 Word 文档图标；链接为 PDF 文件时，后面显示 PDF 文档图标。

根据这一思路，可以编写一个简化的示例代码，如下所示。

① 链接是 <http://ishare.iask.sina.com.cn/>。

- w3cplus.com ☞
- home 合
- Word文档 📄
- Powerpoint文档 📄
- Excel文档 📄
- HTML文档 📄
- PDF文档 📄
- JPG图片文档 📄
- GIF图片文档 📄
- PNG图片文档 📄
- Flash文档 📄
- ZIP压缩文档 📄
- RAR压缩文档 📄
- MP3文档 📄
- EXE安装文件 📄
- TXT文本文档 📄
- w3cplus@hotmail.com ☞
- 本地链接 📄

图 2-64 创建个性化链接

```

<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title> 创建个性化链接样式 </title>
  <style type="text/css">
    a {
      font-size: 16px;
      text-decoration: none;
      padding-right: 20px;
      display: inline-block;
      margin-bottom: 5px;
      background: url(a.png) no-repeat 100% -327px;
    }
    /* 匹配所有外部链接 */
    a[href^="http://"]{
      background-position: 100% -409px;
    }
    /* 匹配首页 */
    a[href="home"]{
      background-position: 100% -382px;
    }
    /* 匹配 .doc 文档 */
    a[href$=".doc"]{
      background-position: 100% 0;
    }
    /* 匹配 .ppt 文档 */
    a[href$=".ppt"]{
      background-position: 100% -50px;
    }
    /* 匹配 .xls 文档 */
    a[href$=".xls"]{

```



```

    background-position:100% -75px;
}
/* 匹配 .html 文档 */
a[href$=".html"]{
    background-position:100% -100px;
}
/* 匹配 .pdf 文档 */
a[href$=".pdf"]{
    background-position:100% -125px;
}
/* 匹配 .jpg, .gif, .png 图片 */
a[href$=".jpg"],
a[href$=".gif"],
a[href$=".png"]{
    background-position:100% -172px;
}
/* 匹配 .swf 文档 */
a[href$=".swf"]{
    background-position:100% -193px;
}
/* 匹配 .zip, .rar 压缩文档 */
a[href$=".zip"],
a[href$=".rar"]{
    background-position:100% -217px;
}
/* 匹配 .mp3 文档 */
a[href$=".mp3"]{
    background-position:100% -245px;
}
/* 匹配 .exe 安装文件 */
a[href$=".exe"]{
    background-position:100% -273px;
}
/* 匹配 .txt 文本文档 */
a[href$=".txt"]{
    background-position:100% -298px;
}
/* 匹配邮箱地址 */
a[href^="mailto"]{
    background-position:100% -350px;
}
</style>
</head>
<body>
<ul>
<li><a href="http://www.w3cplus.com">w3cplus.com</a></li>
<li><a href="home">home</a></li>
<li><a href="a.doc">Word 文档 </a></li>
<li><a href="b.ppt">Powerpoint 文档 </a></li>
<li><a href="c.xls">Excel 文档 </a></li>
<li><a href="d.html">HTML 文档 </a></li>

```

```
<li><a href="e.pdf">PDF 文档 </a></li>
<li><a href="f.jpg">JPG 图片文档 </a></li>
<li><a href="f.gif">GIF 图片文档 </a></li>
<li><a href="f.png">PNG 图片文档 </a></li>
<li><a href="g.swf">Flash 文档 </a></li>
<li><a href="h.zip">ZIP 压缩文档 </a></li>
<li><a href="h.rar">RAR 压缩文档 </a></li>
<li><a href="j.mp3">MP3 文档 </a></li>
<li><a href="h.exe">EXE 安装文件 </a></li>
<li><a href="a.txt">TXT 文本文档 </a></li>
<li><a href="mailto:w3cplus@hotmail.com">w3cplus@hotmail.com</a></li>
<li><a href="#">本地链接 </a></li>
</ul>
</body>
</html>
```

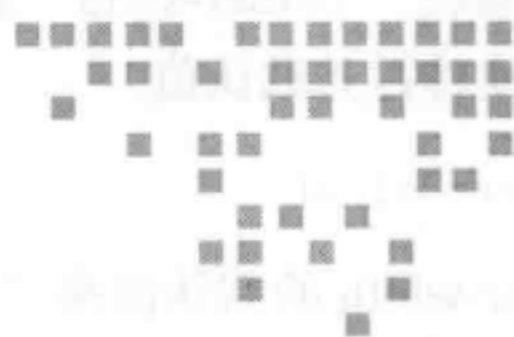
由于 IE 6 浏览器不被支持 CSS3 属性选择器，为了实现图 2-64 效果，早期的 Web 设计师们会借助于 JavaScript 脚本，或者通过古老的办法给不同文档类型元素添加不同的类名，例如给外部链接添加类名“external”，给 Word 文档添加类名“word”。



注意 随着 IE 6 逐渐接近死亡，可以大胆使用 CSS3 属性选择器来制作，或者你是一个完美的追求者，可以考虑优雅降级方案来处理。

2.12 本章小结

本章主要向大家介绍了 CSS3 核心部分中的选择器。首先介绍 CSS3 选择器的优势，然后分别详细介绍了基本选择器、层次选择器、伪类选择器、属性选择器。希望大家学习之后对 CSS3 选择器有一个更详细的了解，为后面的学习打下坚实的基础。



CSS3 边框

提到边框，大家首先想到的是 CSS 的 border 属性。不错，border 属性是 CSS 中盒模型基础属性之一。在 CSS3 中，关于边框的运用会有什么样的不同之处呢？又将如何使用呢？本章我们带着这些问题开始我们的 CSS3 边框之旅。

3.1 CSS3 边框简介

border 属性在 CSS1 中就已经定义了，使用它可以设置元素的边框风格，例如设置不同的边框颜色以及粗细。在详细介绍 CSS3 边框运用之前，先简单回顾边框属性。

3.1.1 边框的基本属性

CSS1 和 CSS2 中的边框属性其实很简单，其主要包括三个类型值。

❑ border-width: 设置元素边框的粗细。

❑ border-color: 设置元素边框的颜色。

❑ border-style: 设置元素边框的类型。

在实际中可以将上面三个属性合并在一起，其缩写的语法：

```
border: border-width border-style border-color;
```

缩写后的每个属性之间使用空格隔开，而且它们之间没有先后顺序，可这里三个值中唯一需要的值是“border-style”，因此，要是这样写边框样式将会没有任何效果。

```
.elm {border:3px red}
```

此时浏览器将“border-style”解析成为“none”。要是这样设置，这个时候元素的边框是实线，粗线将是其默认值。

```
.elm{border:solid}
```

边框 border-width 的默认值是“medium”（大约等于 3 ~ 4px）；border-color 的默认色就是字体的颜色。

在 Web 实际制作之中，时常只为了方便使用，CSS 中的 border 可以给不同的边设置不同的风格，其也遵守“TRBL”原则（Top/Right/Bottom/Left），例如单独写边框类型。

```
border-top-style:/* 设置元素顶部边框类型 */
border-right-style:/* 设置元素右边边框类型 */
border-bottom-style:/* 设置元素底部边框类型 */
border-left-style:/* 设置元素左边边框类型 */
```

上面是边框类型的扩展写法，同样的道理，border-color 和 border-width 也可以像上面一样使用。除了上面的写法之外，还有一种简写形式。

```
border-style: solid;
/* 一个值时，表示四条边都 solid 类型 */
border-style: solid dotted;
/* 两个值时，第一个值表示元素上下边框类型，第二值表示左右边框类型 */
border-style:solid dotted dashed;
/* 三个值时，第一个值表示元素顶边的类型，第二个值表示左右边框类型，第三个值表示底部边框类型 */
border-style: solid dotted dashed inset;
/* 四个值时，第一个值表示元素顶边的类型，第二个值表示元素右边框类型，第三个值表示元素底边的类型，
第四个值表示元素左边框类型 */
```

同样的原理，border-color 和 border-width 具有一样的使用方法。如果只需要设置元素某部分具有边框效果，我们可以合成起来。

```
li {
  border: 1px solid red;
  border-width: 1px 0;
}
```

仅两行代码就表达出元素 li 顶部和底部都有一条 1px 的红色实线。这样方便维护代码，并且提升 CSS 性能。

3.1.2 边框的类型

CSS 中使用 border-style 为元素 border 定义边框类型，常见的有实线“solid”、虚线“dashed”、点状线“dotted”等。下面一起看看 CSS 中 border-style 的几种类型效果，如表 3-1 所示。

表 3-1 border-style 值列表

属性值	功能描述	示例代码	效果
none	定义无边框	.elm {border:none;}	none
hidden	与“none”相同。不过应用于表时除外，对于表，hidden 用于解决边框冲突	.elm{border:hidden;}	hidden
dotted	定义点状边框	.elm{border:3px dotted red ;}	dotted
dashed	定义虚线边框	.elm{border:3px dashed red;}	dashed
solid	定义实线边框	.elm{border:3px solid red;}	solid
double	定义双线。双线的宽度等于 border-width 的值	.elm{border:3px double red;}	double
groove	定义 3D 凹槽边框，其效果取决于 border-color 的值	.elm{border:3px groove red;}	groove
ridge	定义 3D 垄状边框，其效果取决于 border-color 的值	.elm{border:3px ridge red;}	ridge
inset	定义 3D inset 边框，其效果取决于 border-color 的值	.elm{border:8px inset red;}	inset
outset	定义 3D outset 边框，其效果取决于 border-color 的值	.elm{border:8px outset red;}	outset
inherit	规定应该从父元素继承边框样式。部分浏览器不支持这个属性值		

上面 11 个值在各浏览器下呈现的效果均有差异，其中最不可预测的边框样式是 double。它定义为两条线的宽度加上这两条线之间的空间等于 border-width 值。而 dotted、dashed、outset 和 inset 在不同的浏览器下也无法保证一致，如图 3-1 所示。

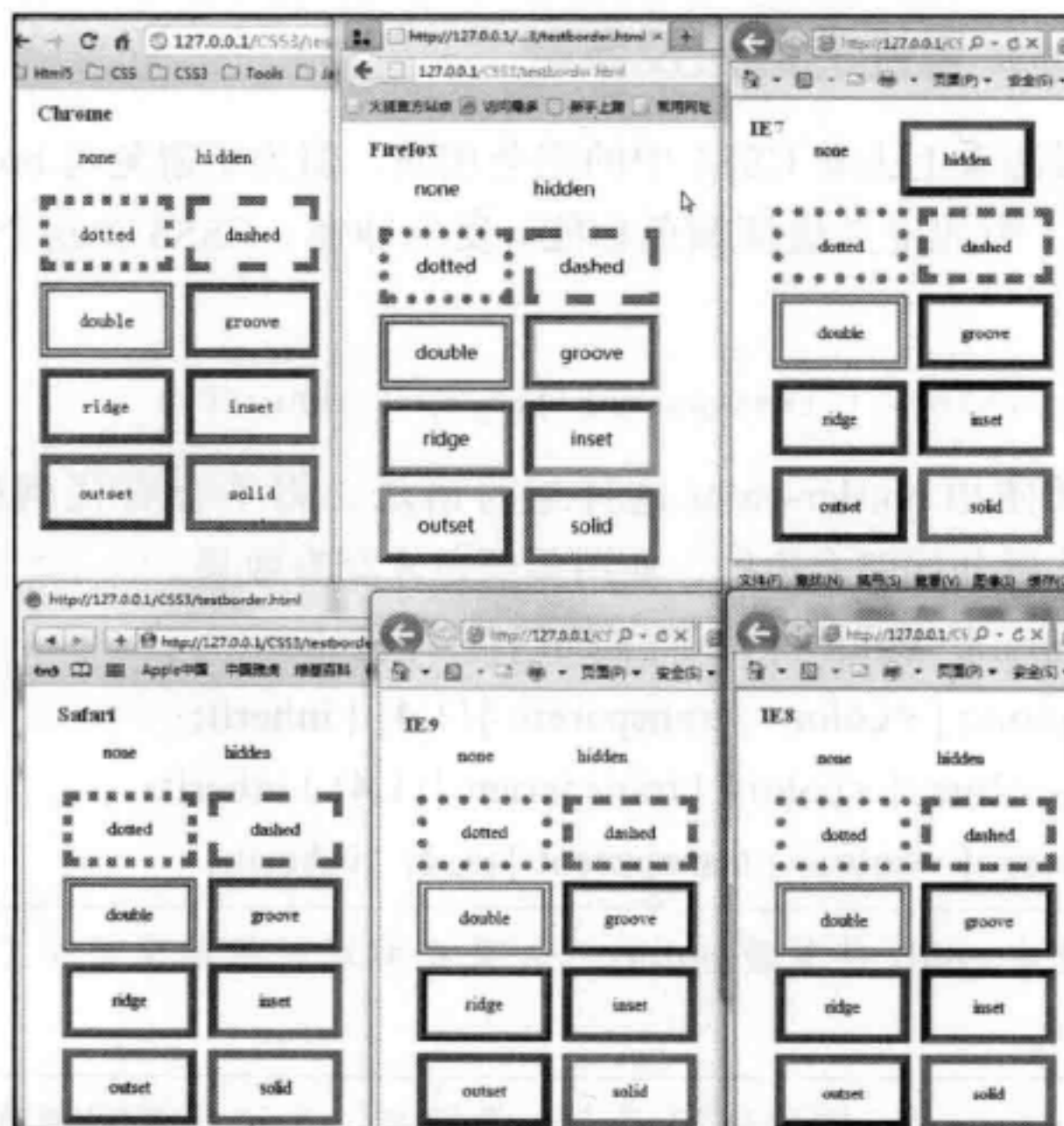


图 3-1 border-style 各浏览器渲染效果



注意 图 3-1 中 IE 7 和 IE 8 未使用原生 IE 测试，而是使用了 IE 9 自带的 IE 7、IE 8 进行的测试。

3.1.3 谁在使用 CSS3 边框

CSS3 增强的边框样式具有强大的生命力，灵活使用这些属性可以设计很多优美精巧的 UI 界面效果。这些属性谁在使用呢？

- ❑ border-color 受制于浏览器兼容性，至今在项目中使用该属性的项目几乎不存在。
- ❑ border-image 浏览器的支持度强一些，但运用在项目中仅存在一些前端爱好者的 blog 中。
- ❑ border-radius 得到浏览器的强大支持，在互联网上随处可见。
- ❑ box-shadow，目前在 Web 页面上 CSS3 的盒子阴影特性应用非常广泛。

3.2 CSS3 边框颜色属性

border-color 属性早在 CSS1 中就已经定义了。不过，CSS3 增强了这个属性的功能，使用它可以为元素边框设置更多的颜色，从而方便 Web 设计人员设计出更为绚丽的边框效果，例如渐变的边框效果、多颜色的边框效果等。

3.2.1 border-color 属性的语法及参数

border-color 的语法看上去和 CSS1 中的完全相同，但为了避免与 border-color 属性的原功能（也就是在 CSS1 中的定义边框颜色功能）发生冲突，CSS3 在这个属性上做出一定的修改。语法如下。

```
border-color: [ <color> | transparent ]{1,4} | inherit
```

换句话说，如果使用 border-color 这种缩写语法，将不会有任何效果，必须将这个 border-color 标准写法拆分成四个边框，使用多颜色才会有效果。

- ❑ border-top-colors: [<color> | transparent]{1,4} | inherit;
- ❑ border-right-colors: [<color> | transparent]{1,4} | inherit;
- ❑ border-bottom-colors: [<color> | transparent]{1,4} | inherit;
- ❑ border-left-colors: [<color> | transparent]{1,4} | inherit;



注意 以上四个属性中 color 是复数 colors，如果在书写过程中要是少了 (s) 字母是错误的，没有任何反应。

由于 CSS3 的 border-color 属性还没成为标准规范，为了让不同浏览器能渲染正常，有

必要加上前缀，如表 3-2 所示。

表 3-2 不同浏览器前缀

浏览器分类	浏览器	私有属性的前端缀
Gecko 引擎内核的浏览器	Mozilla (常指 Firefox 浏览器)	-moz-
Presto 引擎内核的浏览器	Opera	-o-
KHTML 引擎内核的浏览器	Konqueror	-khtml-
Trident 引擎内核的浏览器	Internet Explorer	-ms-

有些情况中，为了使用 CSS3 属性，可能必须添加 4 行代码或者更多行。例如，为了让 border-color 在 Firefox 浏览器下正常，需要加上前缀“-moz-”。

```
-moz-border-top-colors: #111 #222 #333 #444 #555;
-moz-border-right-colors: #111 #222 #333 #444 #555;
-moz-border-bottom-colors: #111 #222 #333 #444 #555;
-moz-border-left-colors: #111 #222 #333 #444 #555;
```

现在的规范还不是最终版本，在执行中还会有一些漏洞。因此，执行这些功能时，使用供应商前缀来提供数值，并且使用无前缀声明来提供每个属性的永久版本。当规范成为最终版本且经过完善后，浏览器前缀将被取消。例如：

```
-moz-border-top-colors: #111 #222 #333 #444 #555;
-moz-border-right-colors: #111 #222 #333 #444 #555;
-moz-border-bottom-colors: #111 #222 #333 #444 #555;
-moz-border-left-colors: #111 #222 #333 #444 #555;

-webkit-border-top-colors: #111 #222 #333 #444 #555;
-webkit-border-right-colors: #111 #222 #333 #444 #555;
-webkit-border-bottom-colors: #111 #222 #333 #444 #555;
-webkit-border-left-colors: #111 #222 #333 #444 #555;

-ms-border-top-colors: #111 #222 #333 #444 #555;
-ms-border-right-colors: #111 #222 #333 #444 #555;
-ms-border-bottom-colors: #111 #222 #333 #444 #555;
-ms-border-left-colors: #111 #222 #333 #444 #555;

-o-border-top-colors: #111 #222 #333 #444 #555;
-o-border-right-colors: #111 #222 #333 #444 #555;
-o-border-bottom-colors: #111 #222 #333 #444 #555;
-o-border-left-colors: #111 #222 #333 #444 #555;

border-top-colors: #111 #222 #333 #444 #555;
border-right-colors: #111 #222 #333 #444 #555;
border-bottom-colors: #111 #222 #333 #444 #555;
border-left-colors: #111 #222 #333 #444 #555;
```

即使用这些前缀来维护代码似乎需要很多工作，现在使用 CSS3，仍然是利大于弊。虽然需要改变一些前缀属性来修改设计元素，相对于通过图形软件更改背景图像或处理那些

其他标记和 hack 脚本，维护基于 CSS3 的设计要容易一些。

Lea Verou 制作了一个插件 -prefix-free^①，使用这个插件，大家在平时的开发中就可以略去浏览器的前缀，从而节约大家的开发与维护成本。



注意 如无特别注明，后面涉及的 CSS3 属性，都需加上各浏览器前缀。

border-color 属性的参数其实很简单，就是颜色值 < color>，可以为任意合法的颜色值或者颜色值列表。当 border-color 只设置一个颜色值时，效果和 CSS1 中的 border-color 效果一样。只有设置了 n 个颜色，并且边框宽度也为 n 像素，就可以使用 CSS3 的 border-color 属性设置 n 个颜色，每种颜色显示 1 像素的宽度，如果宽度值大于颜色数量的值，最后一种颜色用于显示剩下的宽度。例如，元素的边框设置为 20px，而元素的边框颜色只设置了 10 个，剩下的 10px 宽度都将显示最后一种颜色，如图 3-2 所示。

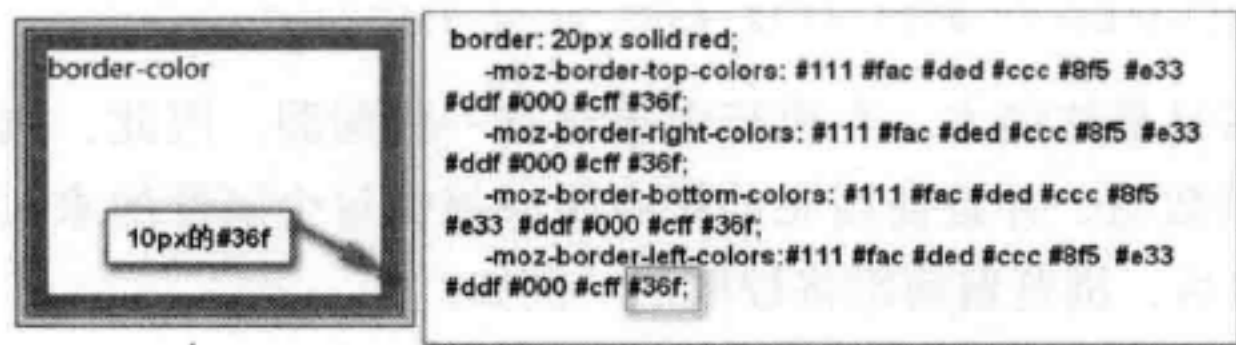


图 3-2 border-color 颜色值设置效果

3.2.2 浏览器兼容性

CSS3 的“border-color”属性浏览器兼容性虽然功能强大，但到写这本书的时候为止，仅有 Firefox 3.0 以及其以上的版本支持，而且还不是标准写法（如表 3-3 所示）。

表 3-3 border-color 浏览器兼容性

属性名					
border-color	x	3.0 + √	x	x	x

CSS3 的 border-color 能帮 Web 设计师制作渐变、内阴影、外阴影等绚丽的边框效果，但是目前为止，仅有 Firefox 3.0 以及其以上版本的浏览器支持，而且还不是标准语法，仅是 Firefox 浏览器自己一个扩展性写法。因此这个属性的使用性并不是很强，大家在实际使用中需要注意。

3.2.3 border-color 属性的优势

在 CSS2 中实现多颜色的边框效果，无外乎两种方法，其一通过添加额外的标签，在每

① 详见 <http://leaverou.github.com/prefixfree/>。

个标签上设置不同的颜色，其二就是通过背景图片来制作。这两种方法和 CSS3 的 border-color 相比都略显弊端，第一种多了标签，使结构冗余，第二种方法背景图片不好维护，特别是在改变边框颜色之时更是麻烦。

有一篇博文^①介绍了使用一个 HTML 标签元素，通过“::before”和“::after”伪元素使用背景色和边框颜色来模拟一个六色边框的效果，这种方法对多颜色受到限制，最多只能制作六种颜色。



注意 除了这些方法，CSS3 中还有 box-shadow 也能实现，详细参考后面介绍 box-shadow 的章节。

3.2.4 实战体验：立体渐变边框效果

在这个案例中，使用 CSS3 的 border-color 属性制作一个渐变立体效果的边框，如图 3-3 所示。

制作这个效果的设计思路很简单，使用 border-color 属性，将颜色从浅到深依次叠加起来，营造出一种立体渐变的边框效果。

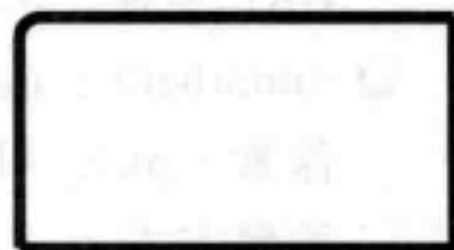


图 3-3 border-color 制作立体渐变边框 (Firefox)

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>border-color 制作立体渐变边框效果</title>
  <style type="text/css">
    div{
      width: 200px;
      height: 100px;
      border: 10px solid transparent;
      border-radius: 15px 0 15px 0;
      -moz-border-top-colors:#a0a #909 #808 #707 #606 #505 #404 #303;
      -moz-border-right-colors:#a0a #909 #808 #707 #606 #505 #404 #303;
      -moz-border-bottom-colors:#a0a #909 #808 #707 #606 #505 #404 #303;
      -moz-border-left-colors:#a0a #909 #808 #707 #606 #505 #404 #303;
    }
  </style>
</head>
<body>
  <div></div>
</body>
</html>
```

3.3 CSS3 图片边框属性

border-image 效果在 CSS2 中，只有使用背景图片来制作，而且制作过程非常复杂，做

^① 地址为 <http://nicolasgallagher.com/multiple-backgrounds-and-borders-with-css2/>。

完后也很难维护。如今 CSS3 中增添了一个图片边框的属性，能够模拟出 background-image 属性的功能，功能比 background-image 强大，我们可以使用 border-image 属性给任何元素（除 border-collapse 属性值为 collapse 的 table 元素之外）设置图片效果边框，还可以使用这个来制作圆角按钮效果、渐变的 Tabs 效果等。

3.3.1 border-image 属性的语法及参数

为了能更好地学习和理解 border-image 这个属性，还是从其最基本的语法入手。

```
border-image: none | <image> [<number> | <percentage>] {1,4} [/ <broder-width> {1,4}] ?[stretch | repeat | round] {0,2}
```

接下来就给大家说说这些参数的含义与使用方法。

- ❑ none：默认值，表示边框无背景图片。
- ❑ <image>：设置背景图片，这跟 background-image 一样，可以使用绝对或相对的 URL 地址，来指定边框的背景图片。
- ❑ <number>：number 是一个数值，用来设置边框或者边框背景图片的大小，其单位是像素（px），可以使用 1～4 个值，表示 4 个方位的值，大家可以参考 border-width 设置方式。
- ❑ <percentage>：percentage 也是用来设置边框或者边框背景图片的大小，跟 number 不同之处是，percentage 使用的是百分比。
- ❑ stretch、repeat、round：这三个属性参数是用来设置边框背景图片的铺放方式，类似于 background-position，其中 stretch 会拉伸边框背景图片、repeat 是会重复边框背景图片、round 是平铺边框背景图片，其中 stretch 为默认值。

border-image 和 background-image 之间有一些类似之处，包括图片的引用和排列方式等。

3.3.2 border-image 属性使用方法

为了更好地理解，暂时把 border-image 在语法的表达形式进行属性的分解阐述（实际应用中是不能分解的，此处只是用来帮助大家更好地理解 border-image 属性）。

- ❑ 引入背景图片：border-image-source。
- ❑ 切割引入背景图片：border-image-slice。
- ❑ 边框图片的宽度：border-image-width。
- ❑ 边框背景图片的排列方式：border-image-repeat。

接下来重点学习 border-image 拆分出来的四个属性。

1. border-image-source

语法：

```
border-image-source: url(image url);
/*image url 可是以边框图片的相对地址，也可以是绝对地址*/
```


`border-image-source` 跟 CSS2 中的 `background-image` 属性相似, 也是通过 `url()` 来调用背景图片, 图片的路径可以是相对地址, 也可以是绝对地址, 当然不想使用背景图片也可以设置为 “`border-image:none`”; 其默认值就是 `none`。

2. border-image-slice

语法:

`border-image-slice:[<number> | <percentage>] {1,4} && fill ?`

`border-image-slice` 是用来分解引入进来的背景图片, 这个参数相对来说比较复杂和特别, 主要表现在以下几点。

1) 取值支持 `<number> | <percentage>`。其中 `number` 是没有单位的, 因为其默认的单位就是像素。除了直接用 `number` 来设置外, 还可以使用百分比值来表示, 即相对于边框背景图片而言的, 例如边框图片的大小是 `300px × 240px`, 取百分比为 25%, 30%, 15%, 20%, 它们实际对应的效果就是剪切了图片的 60px, 90px, 36px, 60px 的四边大小, 如图 3-4 所示。

`border-image-slice` 中的 `number` 或者 `percentage` 都可以取 1 ~ 4 个值, 这个类似于 CSS2 中的 `border-width` 的取值方式, 也遵从 `top`、`right`、`bottom`、`left` 的规则, 如果对这个不太清楚可以参考 CSS2 中的 `border-width` 或者 `padding`、`margin` 等属性的使用方法。

`Fill` 从字面上说就是填充的意思, 如果使用这个关键字时, 图片边界的中间部分将保留下来。默认情况下为空。

2) 剪切的特性 (slice)。在 `border-image` 中 `slice` 是一个关键部分, 也是让人难以理解的部分。如果理解 CSS3 的 `clip` 属性, 再理解 `border-image-slice` 相对会轻松一些。`border-image-slice` 虽然表面上说不是剪切, 但在实际应用中它就是一种纯粹的剪切, 它把通过 `border-image-source` 取到的边框背景图片切成九份, 再像 `background-image` 一样重新布置。

来看一个示例^①, 其剪切的效果如图 3-5 所示。
对应的代码如下所示。

```
div {
    border: 12px double green;
```

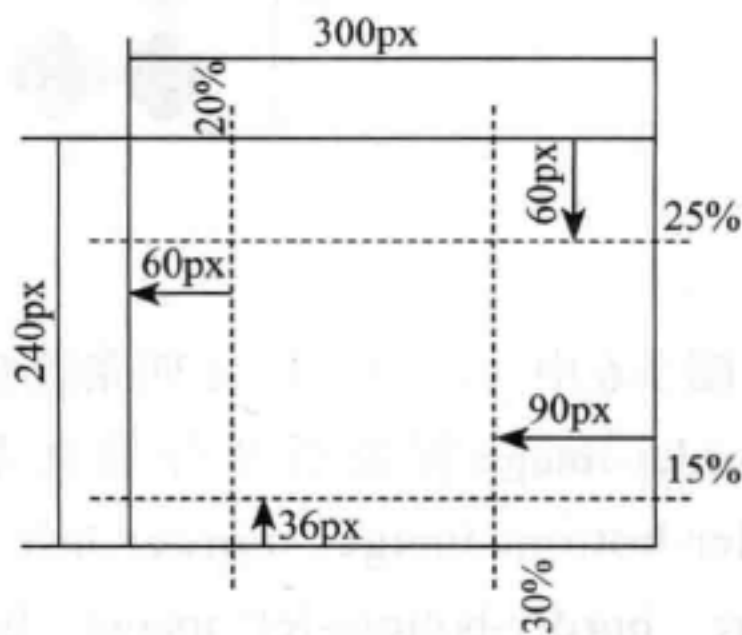


图 3-4 border-image-slice 百分比取值示意图

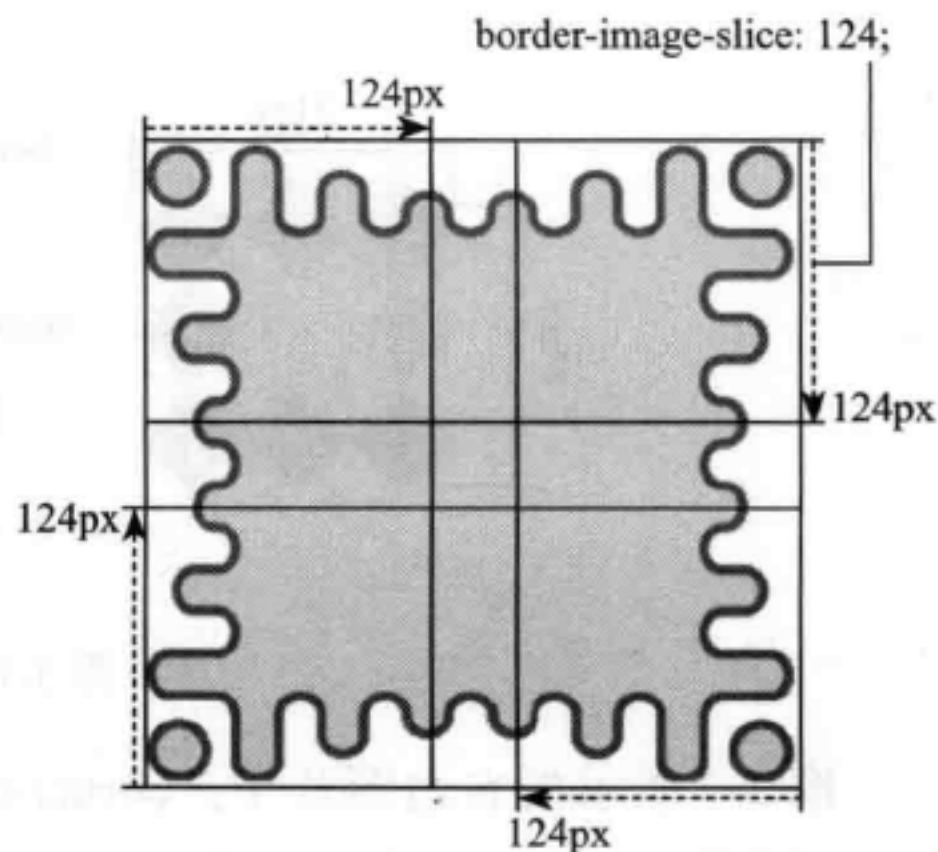


图 3-5 border-image-slice 以 124px 位置切图

① 选自 W3C 官网 (<http://www.w3.org/TR/css3-background/#border-images>)。

```

-moz-border-image: url(../image/border.png) 124;
-webkit-border-image: url(../image/border.png) 124;
-o-border-image: url(../image/border.png) 124;
border-image: url(../image/border.png) 124;
}

```

上面的示意中，它在距边框背景图的 top、right、bottom、left 四边的 124px 分别切了一刀，这样一来就把背景图切成了九个部分，称为“九宫格”。“九宫图”在本文专指由九个方格形成的矩形布局图，正如图 3-5 所示。这样就应用这个“九宫格”来帮助我们了解 border-image 的绘制原理。图 3-6 是来自 W3C 官网的 border-image 背景图，也是一张重要的示意图，因为这张图刚好具有我们所说的“九宫格”(27x3) x (27x3)。

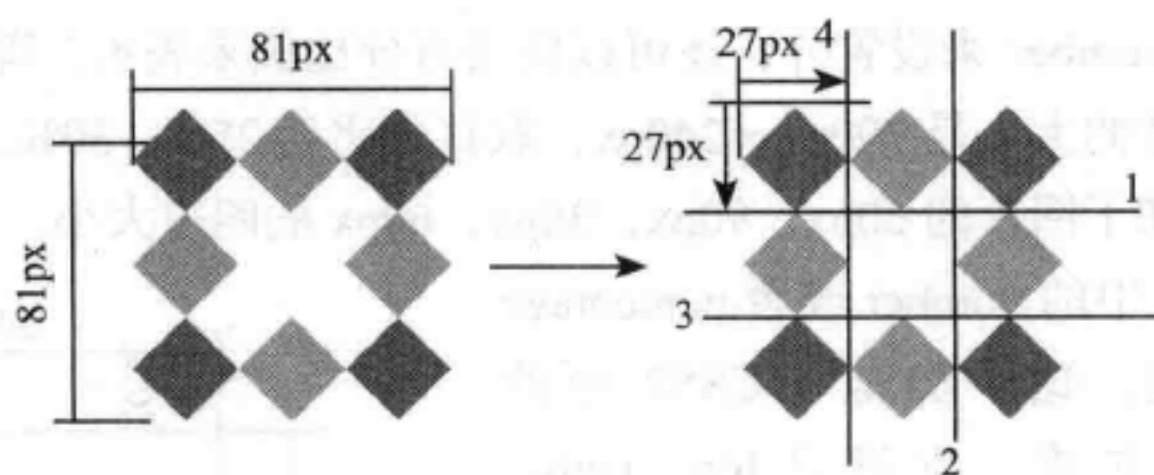


图 3-6 九宫格示意图

图 3-6 中，1、2、3、4 四条蓝色切割线分别在距边框背景图片的 27px 位置切了四刀，将 border-image 背景图片分成九部分。八个边块 border-top-image、border-right-image、border-bottom-image、border-left-image、border-top-right-image、border-bottom-right-image、border-bottom-left-image、border-top-left-image 和最中间的内容区域，如果元素的 border-width 刚好是 27px，则上面所说的九部分正好如图 3-7 所示的对应位置。

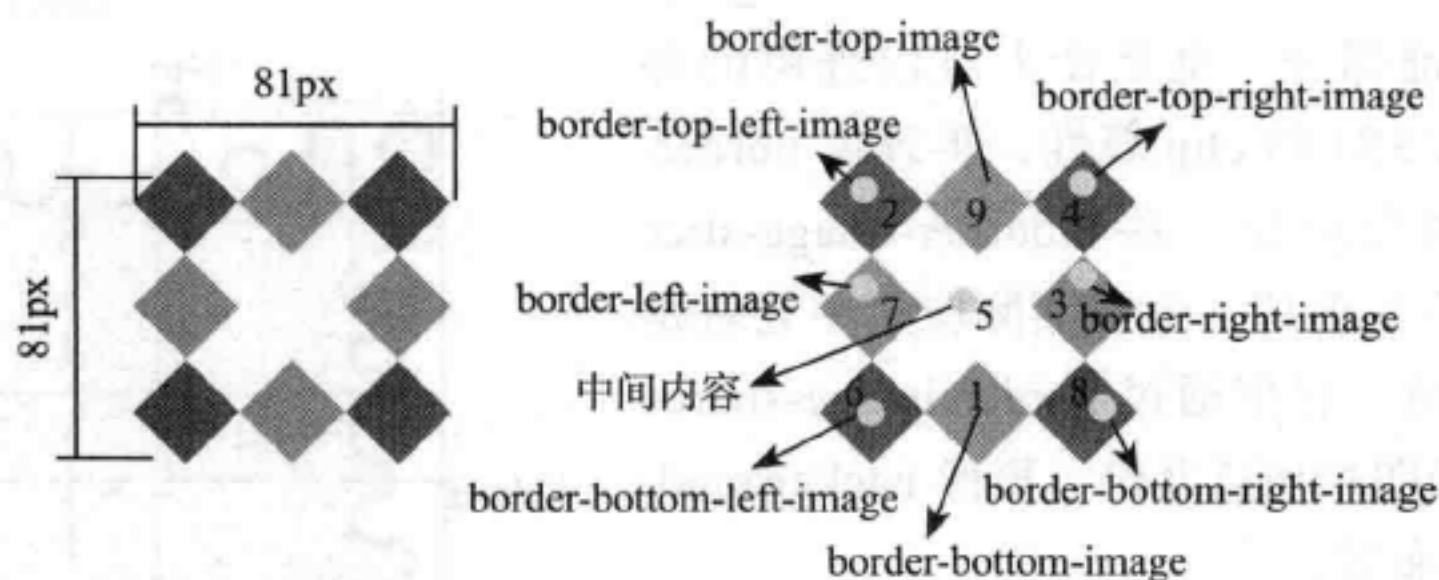


图 3-7 九宫图对应区域

图 3-7 所示的右边图片中，border-top-right-image, border-bottom-right-image, border-bottom-left-image, border-top-left-image 四个边角部分，在 border-image 中是没有任何展示效果的，把这四个部分（图 3-7 中对应的 2、4、6、8 部分）称做盲区；而对应的 border-top-image、border-right-image、border-bottom-image、border-left-image 四个区域在 border-

image 中是属于展示效果区域 (图 3-7 中对应的 1、3、7、9 区域)。

其中上下区域 border-top-image 和 border-bottom-image 区域受到水平方向效果影响。如果是 repeat 则此区域图片会水平重复; 如果是 round 则会水平平铺; 如果是 stretch 则被水平拉伸, 针对这个我们使用案例演示背景图片剪切的方法以及其对应的效果。

假设有一个元素边框背景定义了一个背景图片为 border.png, 然后分别距离边框背景图片顶边 (top)、右边 (right)、底边 (bottom) 和左边 (left) 的 27px 处切一刀, 此时 border-image-slice 的属性值为 (27,27,27,27), 由于四边的值相同, 该属性可以简写为 border-image-slice:27。接下来来看几种不同的切片处理效果。

(1) 水平效果

1) 水平 round 效果。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>border-image 的水平 round 效果</title>
  <style type="text/css">
    .border-image {
      width: 150px;
      height: 100px;
      border: 27px solid;
      -webkit-border-image: url(border.png) 27 round stretch;
      -moz-border-image: url(border.png) 27 round stretch;
      -o-border-image: url(border.png) 27 round stretch;
      -ms-border-image: url(border.png) 27 round stretch;
      border-image: url(border.png) 27 round stretch;
    }
  </style>
</head>
<body>
  <div class="border-image"></div>
</body>
</html>
```

对应的效果如图 3-8 所示。

2) 水平 repeat 效果。

```
.border-image {
  width: 150px;
  height: 100px;
  border: 27px solid;
  -webkit-border-image: url(border.png) 27 repeat stretch;
  -moz-border-image: url(border.png) 27 repeat stretch;
  -o-border-image: url(border.png) 27 repeat stretch;
  -ms-border-image: url(border.png) 27 repeat stretch;
  border-image: url(border.png) 27 repeat stretch;
}
```

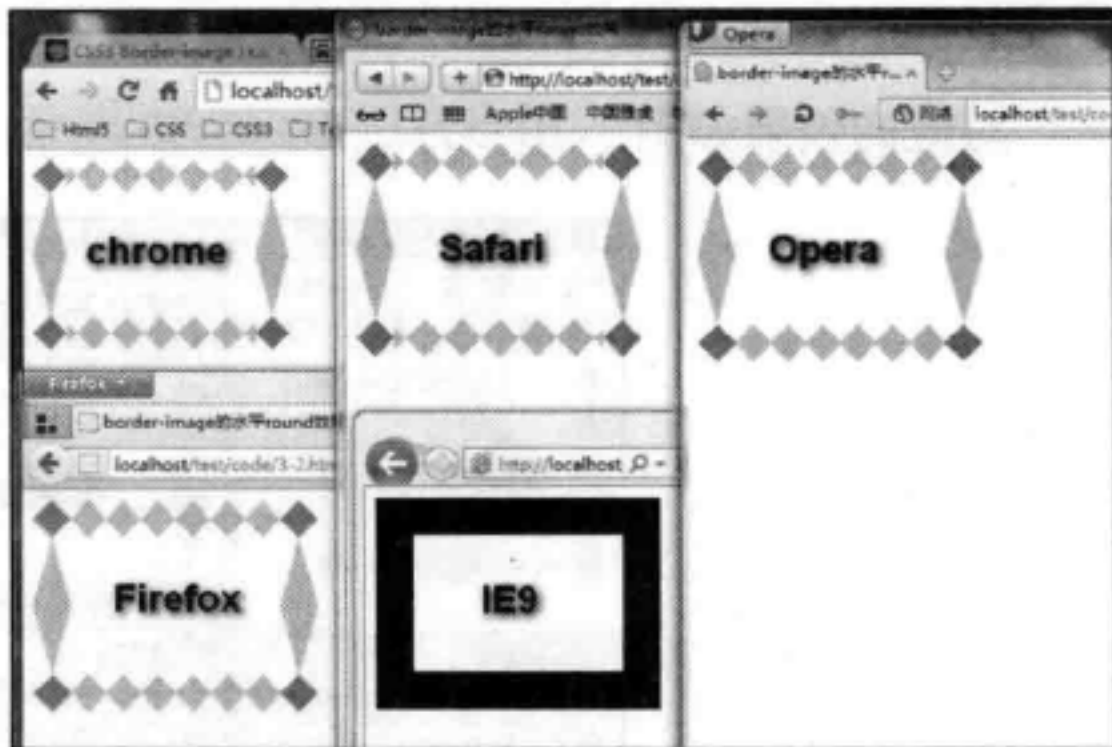


图 3-8 各浏览器下水平 round 演示效果

对应的效果如图 3-9 所示。



图 3-9 各浏览器下水平 repeat 演示效果

3) 水平 stretch 效果。

```
.border-image {
    width: 150px;
    height: 100px;
    border: 27px solid;
    -webkit-border-image: url(border.png) 27 stretch stretch;
    -moz-border-image: url(border.png) 27 stretch stretch;
    -o-border-image: url(border.png) 27 stretch stretch;
    -ms-border-image: url(border.png) 27 stretch stretch;
    border-image: url(border.png) 27 stretch stretch;
}
```

对应的效果如图 3-10 所示。

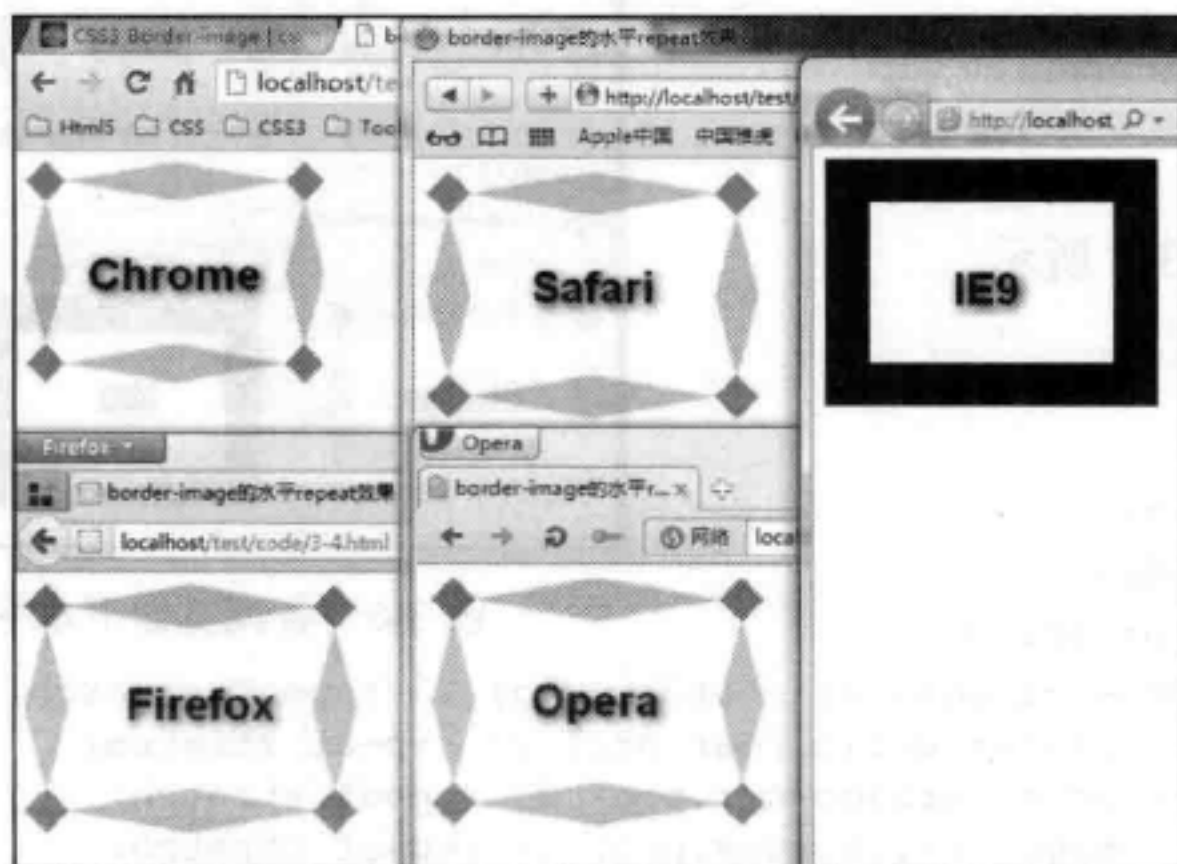


图 3-10 各浏览器下水平 stretch 演示效果

以上分别演示了 border-top-image、border-bottom-image 的 round、repeat 和 stretch 三种效果，从各浏览器下的效果对比图，很容易看出 border-image 在各浏览器下渲染的效果并不一致。在此，只想通过这几个效果来告诉大家 border-top-image 和 border-bottom-image 作用方向，以及对应的 round、repeat 和 stretch 各自会产生何种效果。

(2) 垂直效果

通过前面学习，了解了 border-top-image 和 border-bottom-image 作用区域仅在水平方向，并不会影响垂直方向的效果。由此可以想象，border-right-image 和 border-left-image，只会作用在垂直方向，而且其同样具有 round、repeat、stretch 三种效果。接下来为了验证我们的猜想，一起来看看 border-image 垂直方向的作用效果。

1) 垂直 round 效果。

```
.border-image {
    width: 150px;
    height: 100px;
    border: 27px solid;
    -webkit-border-image: url(border.png) 27 stretch round;
    -moz-border-image: url(border.png) 27 stretch round;
    -o-border-image: url(border.png) 27 stretch round;
    -ms-border-image: url(border.png) 27 stretch round;
    border-image: url(border.png) 27 stretch round;
}
```

对应的效果如图 3-11 所示。

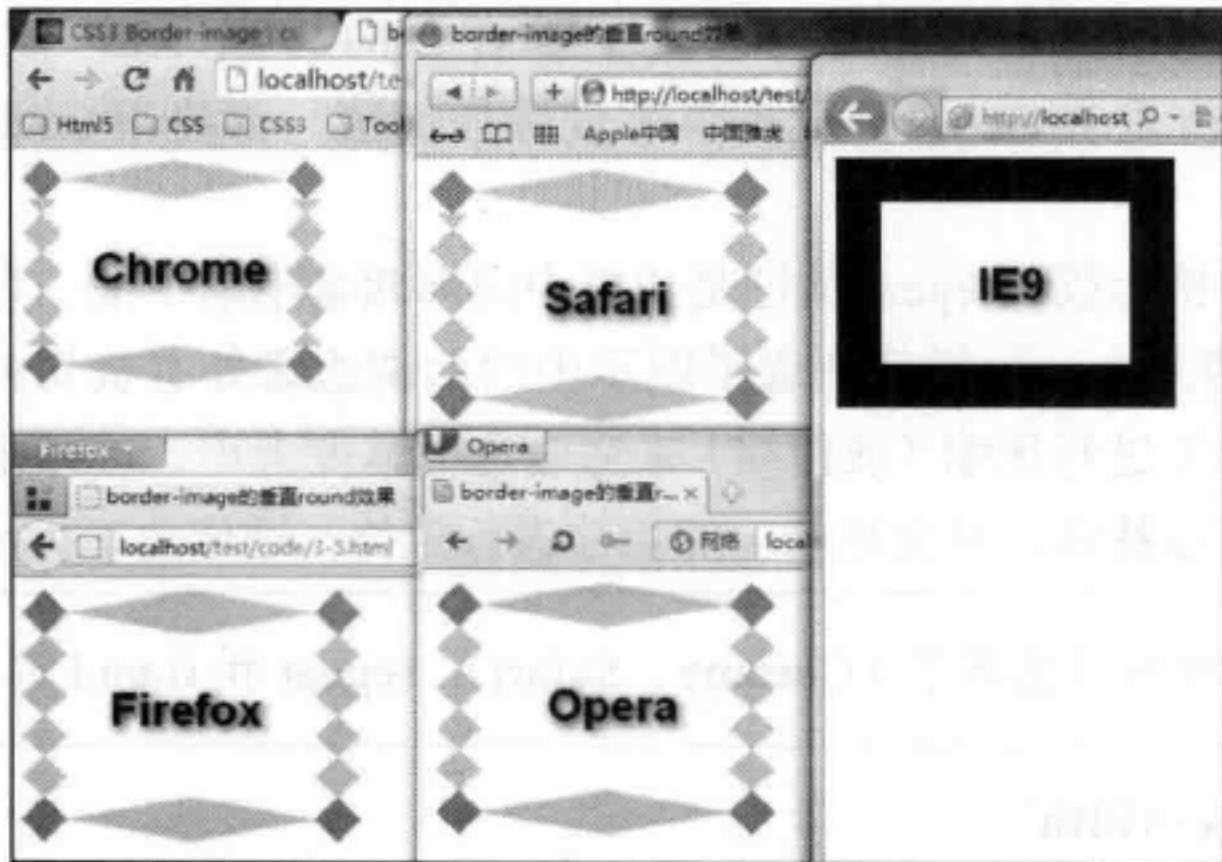


图 3-11 各浏览器下垂直 round 的演示效果

2) 垂直 repeat 效果。

```
.border-image {
    width: 150px;
    height: 100px;
```

```
border: 27px solid;
-webkit-border-image: url(border.png) 27 stretch repeat;
-moz-border-image: url(border.png) 27 stretch repeat;
-o-border-image: url(border.png) 27 stretch repeat;
-ms-border-image: url(border.png) 27 stretch repeat;
border-image: url(border.png) 27 stretch repeat;
}
```

对应的效果如图 3-12 所示。

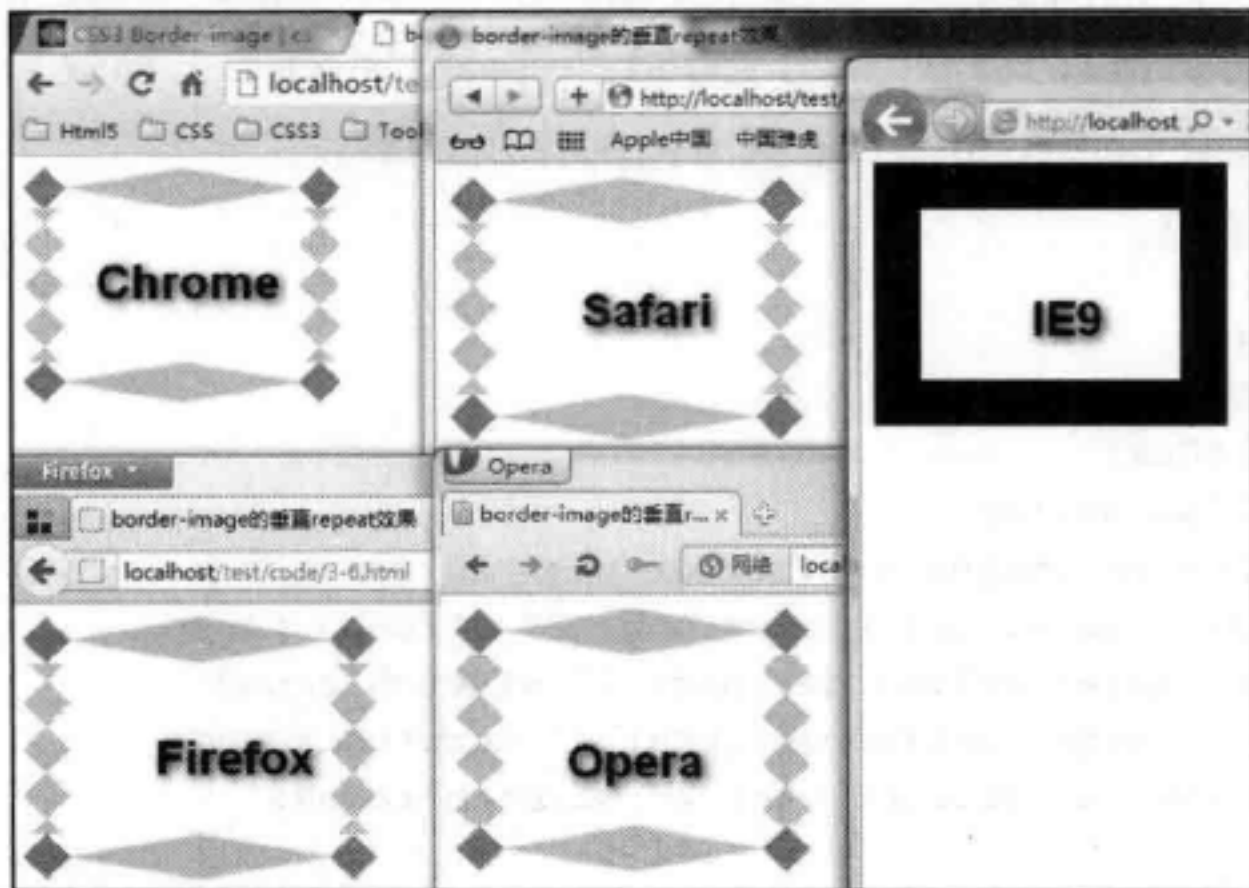


图 3-12 各浏览器下垂直 repeat 的演示效果

其实在演示水平 stretch 效果时也设置了垂直方向的 stretch，前面已说过，border-image-repeat 的默认值是 stretch，因此就算不设置任何值，都将用 stretch 来渲染，具体效果如图 3-10 所示。

通过上面几个示例比较，repeat 属性是边框中间向两端不断平铺，在平铺的过程中保持边框背景图片切片的大小，这样就造成了图示中的两端边缘处有被切的现象。而 round 却会对边框背景图的切片进行压缩（或伸缩）来适应边框宽度大小，进行排列，使其正好显示在区域内。stretch 有点特殊，只会把相应的切片进行拉伸，适应边框大小。



注意 在 Webkit 内核的浏览器下（Chrome、Safari），repeat 和 round 两者效果无区别。

3. border-image-width

语法：

```
boder-image-width: [<length> | <percentage> | <number> | auto] {1,4}
```

用来设置边框背景图片的显示大小，其实也可以理解为 border-width。虽然 W3C 定义了 border-image-width 属性，但各浏览器还是将其视为 border-width 来用，也就是说它和 border-width 的使用方法是一样的。

4. border-image-repeat

语法：

```
border-image-repeat: [stretch | repeat | round] {1,2}
```

用来指定边框背景图片的排列方式，其默认值为 stretch。这个属性设置参数和其他的不一样，border-image-repeat 不遵循 top、right、bottom、left 的方位原则，它只接受两个（或一个）参数值，第一个值表示水平方向的排列方式，第二个值表示垂直方向的排列方式。当只取一个值时，表示水平和垂直方向的排列方式相同。如果你不显式设置任何值时，水平和垂直都会以其默认值 stretch 方式来进行排列。

为了能让大家更好地理解 border-image-repeat 的使用，下面将结合 border-image-slice 一起来看看 round、repeat 和 stretch 的实现原理。

在上面的示例中，使用一个 81px × 81px 的背景图片 border.png，分别在背景图片的顶边、右边、底边和左边的第 27px 处切了四刀，分成九个部分，每个方块的高和宽都是 27px × 27px。其中有四个部分是盲区，不管什么排列方式，这四个区都不变（border-top-right-image、border-bottom-right-image、border-bottom-left-image、border-top-left-image），而 border-top-image 和 border-bottom-image 两部分随着排列方式不同而效果不同，只限于水平方向的排列变化；另外两个 border-right-image 和 border-left-image 只是在垂直方向进行排列；最后中间部分同时在水平和垂直方向平铺，如图 3-13 所示。

前面把 border-image 像 background 一样分解介绍了其相关知识点，但在实际应用中，border-image 各属性必须写在一起，不能分解。下面给大家提供一个正确的速记法。

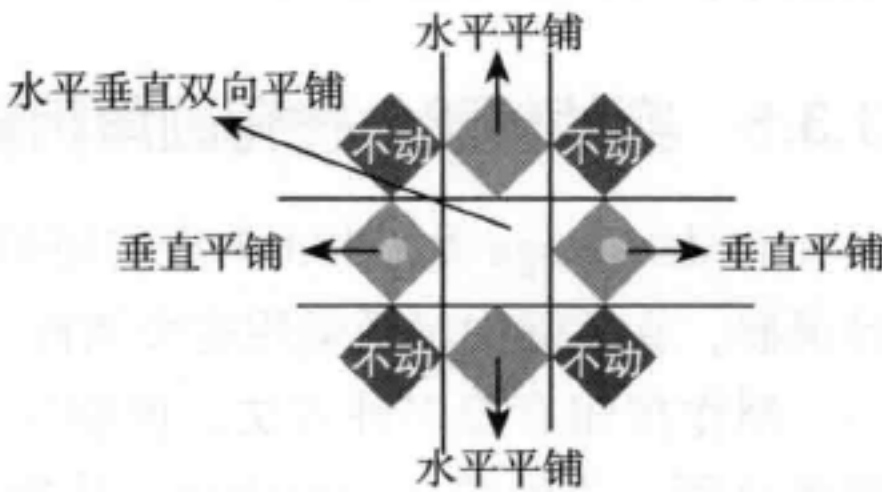


图 3-13 九宫图

```
border-image:<border-image-source> || <border-image-slice> [ /<border-image-width> ] || <border-image-repeat>
```

3.3.3 浏览器兼容性

border-image 是 CSS3 新增的核心属性之一，也是一个非常实用的属性。随着主流浏览器的全面支持，这个属性会更实用。目前使用 border-image 属性，还是需要带上浏览器的私有属性，如表 3-4 所示。

表 3-4 各浏览器对 border-image 的私有属性

	Mozilla Gecko	Webkit	Presto	Konqueror	Internet Explorer
border-image	-moz-	-webkit-	-o-	-khtml-	-ms-

目前 IE 系列并不支持，也没有定义 `-ms-border-image` 的私有属性，其他各主流浏览器的支持情况如表 3-5 所示。

表 3-5 border-image 的浏览器兼容性

属性名					
border-image	x	3.5 + √	3.0 + √	10.5 + √	1.0 + √

3.3.4 border-image 属性的优势

`border-image` 功能强大，但受限于浏览器的支持度，其使用还是受到很大的限制。但相信这个功能将会在未来的 Web 应用中得到广泛的运用，尽展个人的魅力。

以前，给某个元素添加图片边框效果，唯一的办法就是使用背景图片。如果知道元素的尺寸会简单点，使用滑动门技术就可以实现，如果元素的尺寸不定，也就是说元素宽度、高度都自适应，单独使用背景图片还很难实现。在这种情况下就需要添加很多空标签，使用九宫格来填充背景。使用 `border-image` 就轻松多了，只需要一张背景图片可以让某个元素实现图片边框的效果，或者其他效果，如圆角效果、阴影效果等。这样大大提高了开发效率，降低了开发成本。

3.3.5 实战体验：按钮圆角阴影效果

`border-image` 是 CSS3 中很实用的属性，接下来通过几个小案例帮助读者拓展自己的设计灵感，在实际中灵活运用这个属性。

制作按钮有很多种方法，但制作自适应宽度的圆角按钮还是很头痛的。早期使用四个圆角分别定位到按钮的四个角，接着有人使用一张圆角背景图片，通过滑动门技术来制作圆角按钮。随着 `border-radius` 的出现，很多情况下使用这个属性制作圆角按钮。本节介绍用 `border-image` 制作按钮的案例。

首先需要一张图片，当做 `border-image` 的背景图片，如图 3-14 所示。
接下来一起来看案例的实现代码。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>CSS3 的 border-image 制作按钮 </title>
  <style type="text/css">
    .border-image-btn {
      display: inline-block;
      border: 18px solid green;
      border-width: 0 18px;
      border-image: url("button_sprite.png") 0 18 50 18;
      -webkit-border-image: url("button_sprite.png") 0 18 50 18;
```



图 3-14 制作按钮的背景图片


```

-moz-border-image: url("button_sprite.png") 0 18 50 18;
-o-border-image: url("button_sprite.png") 0 18 50 18;
-ms-border-image: url("button_sprite.png") 0 18 50 18;

padding: 13px 10px 17px;
font-size: 16px;
color: #fff;
font-weight: bold;
text-decoration: none;
line-height: 15px;
margin: 10px;
}

.border-image-btn:hover {
border-image: url("button_sprite.png") 50 18 0 18;
-webkit-border-image: url("button_sprite.png") 50 18 0 18;
-moz-border-image: url("button_sprite.png") 50 18 0 18;
-o-border-image: url("button_sprite.png") 50 18 0 18;
-ms-border-image: url("button_sprite.png") 50 18 0 18;

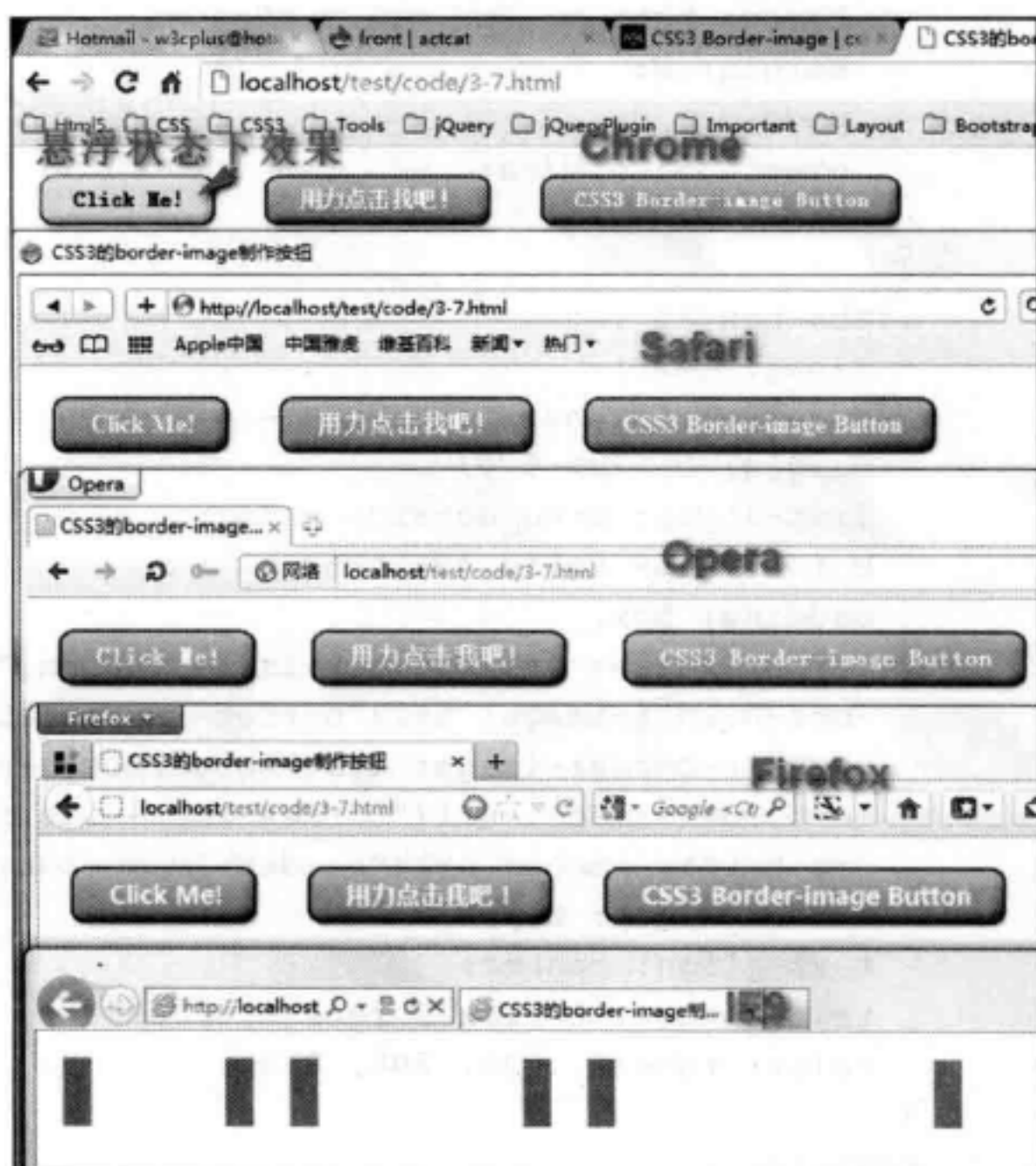
color: #000;
border-color: yellow;
text-decoration: none;
}
</style>
</head>
<body>
<a href="#" class="border-image-btn">Click Me!</a>
<a href="#" class="border-image-btn">用力点击我吧! </a>
<a href="#" class="border-image-btn">CSS3 Border-image Button</a>
</body>
</html>

```

各浏览器下 border-image 制作按钮效果如图 3-15 所示。

从图 3-15 所示的效果中可以看出, border-image 在现代浏览器下得到较好的支持, 唯有 IE 系列不支持 (希望 IE 10 能支持)。接下来简单说一下原理。

这个简单的案例中, 首先采用了一张 40px × 100px 的图片精灵 (如图 3-14 所示) 作为元素的边框背景



☆图 3-15 各浏览器下 border-image 制作按钮效果

图像，然后在距图片顶边 0px 处切第一刀，在距图片右边 18px 处切第二刀，在距图片底边 50px 处切第三刀，在距图片左边 18px 处切第四刀，从而组成九宫格。接下来利用 border-image 的拉伸属性，实现 border-image 制作按钮的默认效果。按钮悬浮状态下，采用相同的办法，只是改变切片的位置来达到一样的效果。这里还有关键一步，按钮的边框宽度只有左右，如果上下也要设置边框宽度，上面的切图就无法达到所需的效果。感兴趣的同学不妨一试。



注意 在 Chrome 浏览器下 border-image 的标准写法写在最后，会造成不可预计的错误效果。

使用 border-image 除了可以制作上面的按钮效果之外，还可以制作 tabs 效果，其原理是一样的，此处不再做过多的重复阐述。但有一个关键处就是 border-image 的背景图片源要制作好，这里采用的背景图像如图 3-16 所示。

一起来看看示例代码。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>CSS3 的 border-image 制作 tabs 效果</title>
  <style type="text/css">
    .tabs-box {
      border-bottom: 3px solid #9eaab6;
      margin: 0;
      padding: 0;
      overflow: hidden;
      zoom: 1;
    }
    .tabs-box li {
      float: left;
      display: inline;
      margin: 0 12px 0 0;
      list-style: none outside none;
      border: 1px solid #9EAAB6;
      padding: 5px;
      border-image: url("border-image-tab.png") 0 5 0 5;
      -moz-border-image: url("border-image-tab.png") 0 5 0 5;
      -webkit-border-image: url("border-image-tab.png") 0 5 0 5;
      -o-border-image: url("border-image-tab.png") 0 5 0 5;
      -ms-border-image: url("border-image-tab.png") 0 5 0 5;
      border-width: 0 5px;
      text-align: center;
      text-shadow: 0 -1px 0 rgba(0,0,0,0.8);
      color: rgba(0, 125, 200, 0.3);
    }
  </style>
</head>
```



图 3-16 border-image 制作 tabs 效果的背景图


```

<body>
  <ul class="tabs-box">
    <li>Home</li>
    <li>CSS3</li>
    <li>Html5</li>
    <li>JavaScript</li>
    <li>jQuery</li>
  </ul>
</body>
</html>

```

示例效果如图 3-17 所示。

使用 border-image 制作 tabs 效果是不是比滑动制作要方便快捷。不过使用 border-image 制作需要掌握三点：1) 源图片制作恰当；2) 动刀切边框背景图片合理；3) 边框宽度配合到位。

比如此例的切图如图 3-18 所示。

接下来看使用 border-image 制作圆角与阴影的示例。和前面两个示例一样，需要制作好的边框背景图，如图 3-19 所示。



☆图 3-17 各浏览器下 border-image 制作 tabs 效果

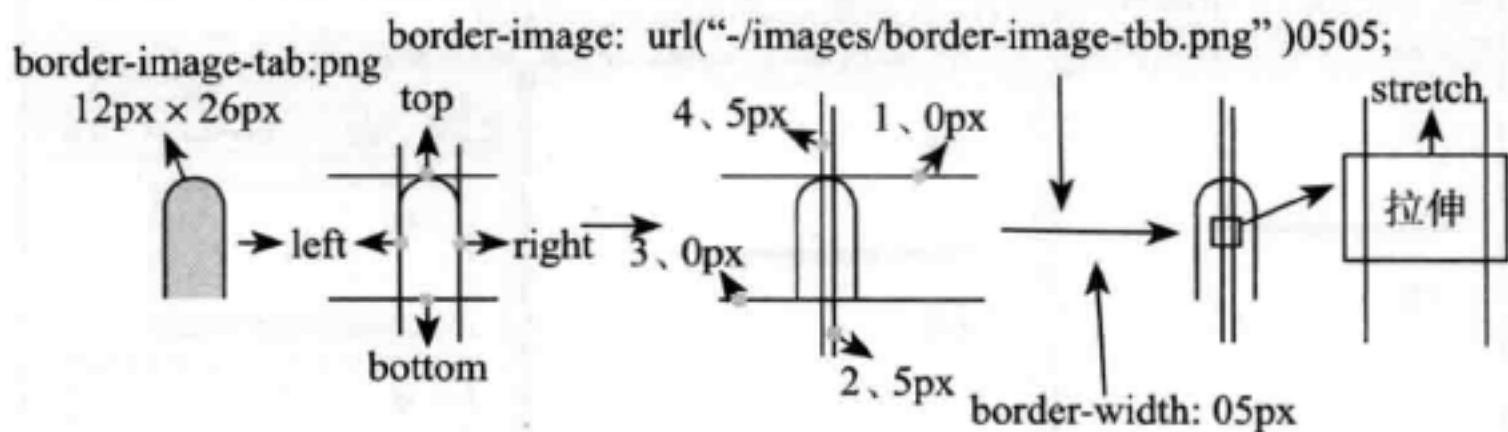


图 3-18 border-image 切图



图 3-19 制作圆角和阴影的边框单背景图

示例代码如下。

```

<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>CSS3 的 border-image 制作圆角和阴影效果</title>
  <style type="text/css">

```

```

.border-image-drop-boxshadow {
width: 150px;
height: 50px;
padding: 10px;
margin: 10px;
border: 1px solid #ccc;
border-width: 7px 7px 16px;
border-image: url("border-image-box-shadow.png") 7 7 16 7;
-moz-border-image: url("border-image-box-shadow.png") 7 7 16 7;
-webkit-border-image: url("border-image-box-shadow.png") 7 7 16 7;
-o-border-image: url("border-image-box-shadow.png") 7 7 16 7;
-ms-border-image: url("border-image-box-shadow.png") 7 7 16 7;
}
.box2{
width: 200px;
height: 100px;
}
</style>
</head>
<body>
<div class="border-image-drop-boxshadow box1"> 小框 </div>
<div class="border-image-drop-boxshadow box2"> 大框 </div>
</body>
</html>

```

效果如图 3-20 所示。

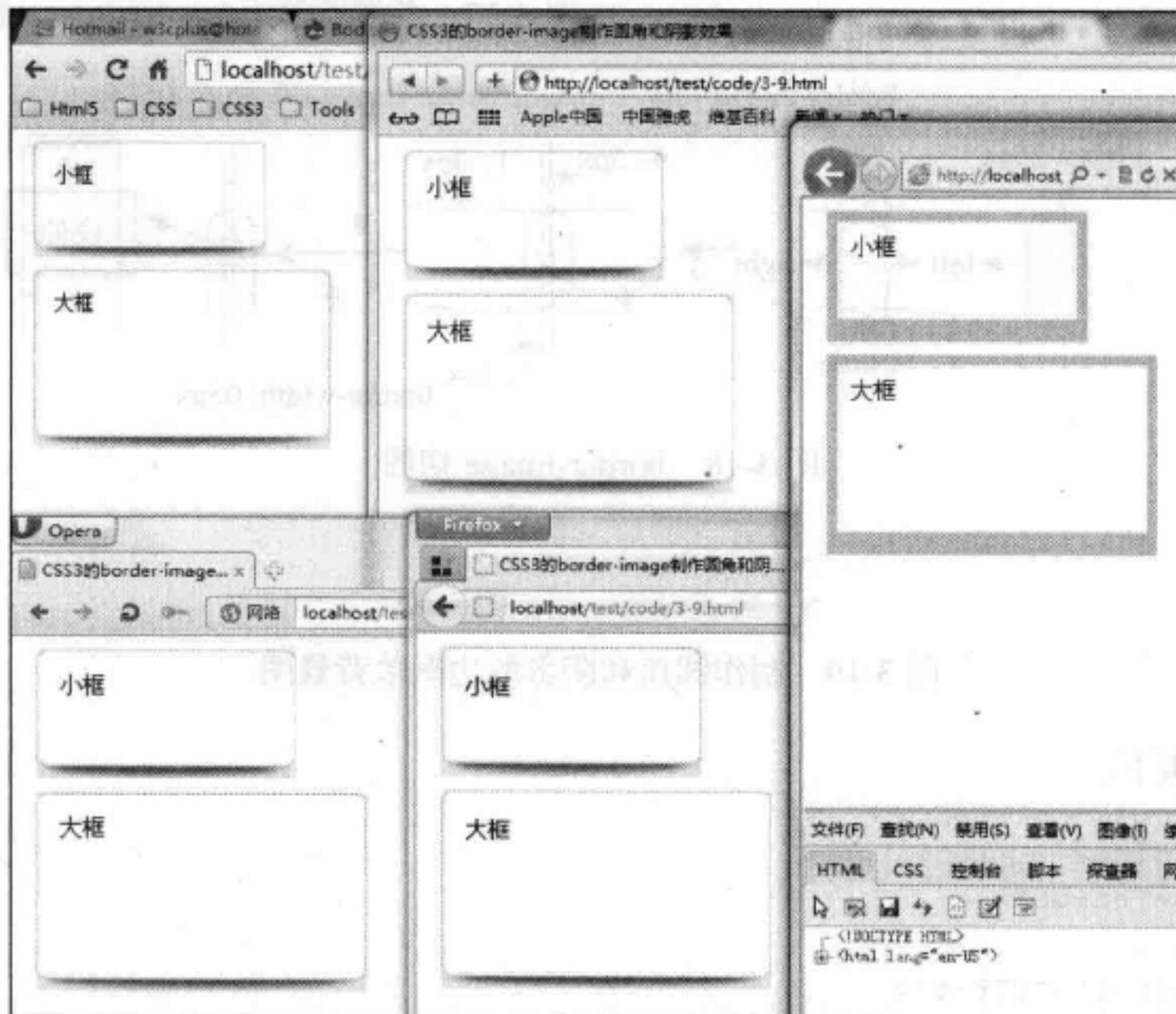


图 3-20 各浏览器下 border-image 制作圆角与阴影效果

以上几个案例可以体现 border-image 在实际应用中非常灵活,可以根据不同需求设计不同的边框背景图,设置不同 border-image-slice 属性值,从而设计各种各样的特殊边框样式,如带有纹理的相框、带有花边的边框等,大家不妨亲自体验一下。

3.4 CSS3 圆角边框属性

在 Web 页面上圆角效果很常见。圆角给页面增添曲线之美,让页面不那么生硬,但是为了设计圆角效果,Web 设计师们要花费更多的时间与精力。

3.4.1 border-radius 属性的语法及参数

CSS3 中专门针对元素的圆角效果增加了一个圆角属性 border-radius。Web 设计师不会为 Web 页面中的圆角效果纠结了。

语法:

```
border-radius: none | <length> {1,4} [/<length>{1,4}] ?
```

border-radius 是一种缩写方法。如果反斜杠符号“/”存在,“/”前面的值是设置元素圆角的水平方向半径,“/”后面的值是设置元素圆角的垂直方向的半径;如果没有“/”,则元素圆角的水平和垂直方向的半径值相等。另外四个值是按照 top-left、top-right、bottom-right 和 bottom-left 顺序来设置的,其主要会有以下四种情形出现。

1) border-radius:<length>{1} 设置一个值, top-left、top-right、bottom-right 和 bottom-left 四个值相等,也就是元素四个圆角效果一样。

2) border-radius:<length>{2} 设置两个值, top-left 等于 bottom-right, 并且取第一个值; top-right 等于 bottom-left, 并且取第二个值。也就是元素的左上角和右下角取第一个值, 右上角和左下角取第二个值。

3) border-radius:<length>{3} 设置三个值, 第一个值设置 top-left, 第二个值设置 top-right 和 bottom-left, 第三个值设置 bottom-right。

4) border-radius:<length>{4} 元素四个圆角取不同的值, 第一个值设置 top-left, 第二个值设置 top-right, 第三个值设置 bottom-right, 最后一个值设置 bottom-left。

border-radius 的属性参数非常简单, 主要包含两个值。

□ none: 默认值, 表示元素没有圆角。

□ <length>: 由浮点数字和单位标识符组成的长度值。不可以是负值。



注意 如果要重置元素没有圆角, 取值 none 并无效果, 需要将元素的 border-radius 取值为 0。

border-radius 和 border 属性一样, 可以将各个角单独拆分出来。这样 border-radius 就

派生出另外四个子属性，而且它们都是先 Y 轴再 X 轴。

□ border-top-left-radius: <length>/<length>; 定义元素左上角圆角。

□ border-top-right-radius: <length>/<length>; 定义元素右上角圆角。

□ border-bottom-right-radius: <length>/<length>; 定义元素右下角圆角。

□ border-bottom-left-radius: <length>/<length>; 定义元素左下角圆角。

上面四个子属性取值和 border-radius 是一样的，只不过水平和垂直方向仅一个值，“/”前面的值为水平方向半径，后面的值为垂直方向半径。如果第二个值省略，元素水平和垂直方向半径，其实就是以“<length>”为半径的四分之一圆。如果任意一个值为“0”，这个角就不是圆角。

由于各浏览器厂商对 border-radius 子属性解析不一致，造成了各浏览器下的 border-radius 属性的派生子属性写法有所区别。

1) Gecko 内核浏览器 (Firefox、Flock 等)。

```
-moz-border-radius-topleft:<length>/<length>;    左上角圆角
-moz-border-radius-topright:<length>/<length>;    右上角圆角
-moz-border-radius-bottomright:<length>/<length>;  右下角圆角
-moz-border-radius-bottomleft:<length>/<length>;   左下角圆角
```

2) Webkit 内核浏览器 (Chrome、Safari 等)。

```
-webkit-border-top-left-radius: <length>/<length>;    左上角圆角
-webkit-border-top-right-radius: <length>/<length>;    右上角圆角
-webkit-border-bottom-right-radius: <length>/<length>;  右下角圆角
-webkit-border-bottom-left-radius: <length>/<length>;   左下角圆角
```

3) Presto 和 Trident 内核浏览器 (Opera、IE 9+ 等)。

```
border-top-left-radius: <length>/<length>;    左上角圆角
border-top-right-radius: <length>/<length>;    右上角圆角
border-bottom-right-radius: <length>/<length>;  右下角圆角
border-bottom-left-radius: <length>/<length>;   左下角圆角
```

border-radius 派生的子属性虽然方便为元素设置指定角的圆角，但为了兼容各浏览器的新老版本写法，不得为样式增加额外的代码。

/*Firefox 浏览器*/

```
-moz-border-radius-topleft: <length>/<length>;    右上角圆角
-moz-border-radius-topright: <length>/<length>;    右上角圆角
-moz-border-radius-bottomright: <length>/<length>;  右下角圆角
-moz-border-radius-bottomleft: <length>/<length>;  左下角圆角
```

/*Chrome 和 Safari 浏览器*/

```
-webkit-border-top-left-radius: <length>/<length>;    左上角圆角
-webkit-border-top-right-radius: <length>/<length>;    右上角圆角
-webkit-border-bottom-right-radius: <length>/<length>;  右下角圆角
-webkit-border-bottom-left-radius: <length>/<length>;  左下角圆角
```

/*Opera、IE 9+、W3C 标准写法*/

<code>border-top-left-radius: <length>/<length>;</code>	左上角圆角
<code>border-top-right-radius: <length>/<length>;</code>	右上角圆角
<code>border-bottom-right-radius: <length>/<length>;</code>	右下角圆角
<code>border-bottom-left-radius: <length>/<length>;</code>	左下角圆角

这样给元素设置单个圆角效果是件非常痛苦的事情,而且难以维护,也容易出错。其实给元素设置单个圆角效果,完全可以借助 `border-radius` 属性的标准写法,只是需要将其其他顶边的圆角半径值设置为 0。例如,只要给元素左上角设置圆角效果。

```
border-radius: 5px 0 0 0; /* 左上角设置圆角 */
```

3.4.2 border-radius 属性使用方法

前面了解了 `border-radius` 的语法,同时知道 `border-radius` 属性可以包含两个参数值,第一个是水平圆角半径值,第二个是垂直圆角半径值,而且两个参数值使用“/”分开,如图 3-21 所示^①。

1. 水平和垂直半径一样

通过一些简单的示例代码进一步加强对 `border-radius` 的理解。

1) border-radius 只设置一个值。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>CSS3 的 border-radius 制作圆角 </title>
  <style type="text/css">
    .border-radius {
      width: 250px;
      height: 100px;
      border: 10px solid orange;
      border-radius: 10px;
    }
  </style>
</head>
<body>
  <div class="border-radius"></div>
</body>
</html>
```

此时,元素四个角都具有圆角,而且圆半径值一样,效果也一样,如图 3-22 所示。

2) border-radius 设置两个值。

```
.border-radius {
  width: 350px;
```

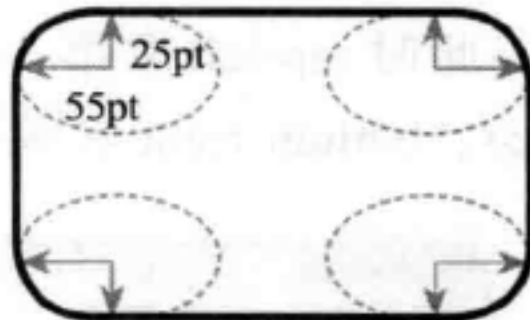


图 3-21 border-radius 的圆角

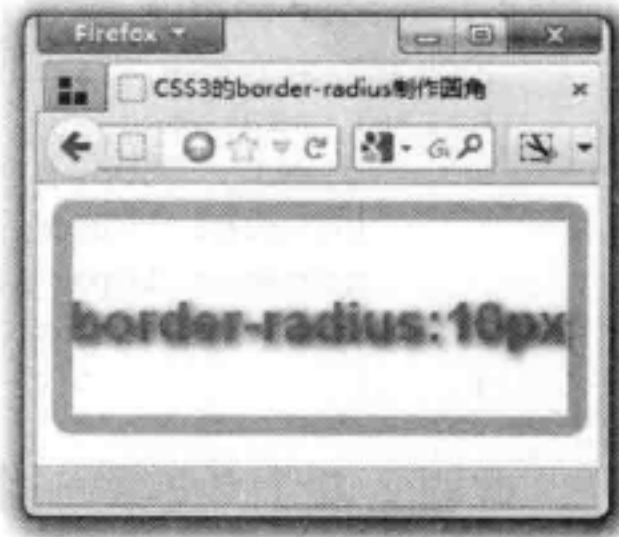


图 3-22 border-radius 取一个值时圆角边框效果

^① 图 3-21 来自于 <http://www.w3.org/TR/css3-background/#border-radius>。

```

height: 100px;
border: 10px solid orange;
border-radius: 10px 30px;
}

```

此时 top-left 等于 bottom-left 并且它们取第一个值 10px；top-right 等于 bottom-right 并且取第二个值 30px。即元素左上角和右下角圆角相同，其圆角半径值为 10px，而元素右上角和左下角圆角相同，其圆角半径值为 30px，如图 3-23 所示。

3) border-radius 设置三个值。

```

.border-radius {
width: 350px;
height: 100px;
border: 10px solid orange;
border-radius: 10px 50px 30px;
}

```

此时 top-left 取第一个值 10px，top-right 和 bottom-left 圆角效果一样，取第二个值 50px，bottom-right 取第三个值 30px，如图 3-24 所示。

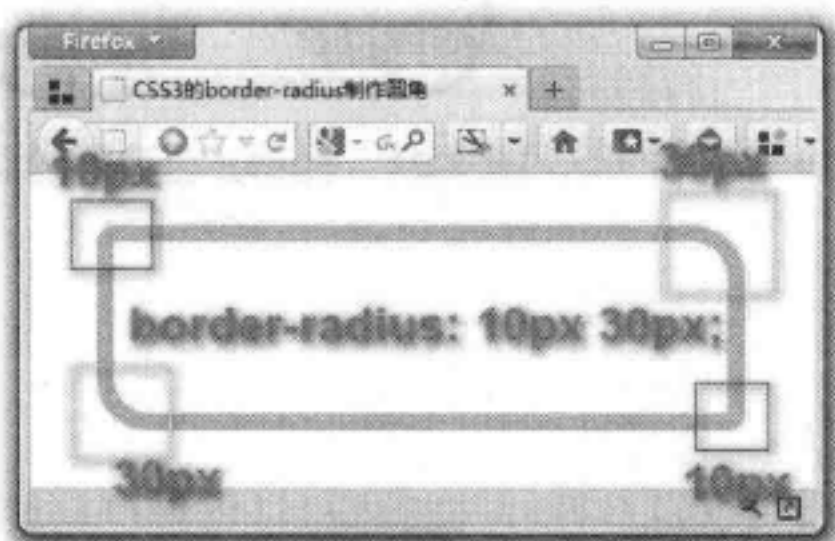


图 3-23 border-radius 取两个值的圆角边框效果

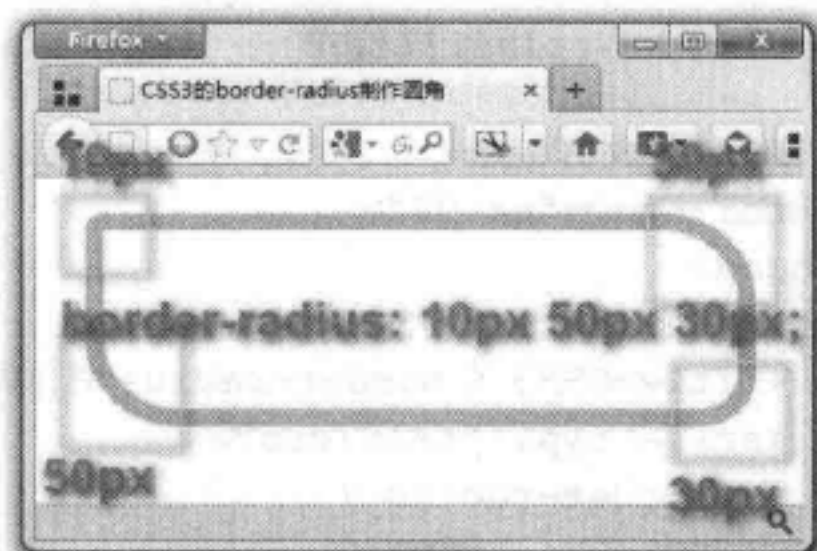


图 3-24 border-radius 取三个值时圆角边框效果

4) border-radius 设置四个值。

```

.border-radius {
width: 350px;
height: 100px;
border: 10px solid orange;
border-radius: 10px 20px 30px 40px;
}

```

此时，左上角取第一个参数值 10px，右上角取第二个参数值 20px，右下角取第三个参数值 30px，左下角取第四个参数值 40px；如果四个值不相同，意味着元素的四个顶角的圆角效果都不一样，如图 3-25 所示。

2. 单独设置水平和垂直半径值

上面展示的是 border-radius 设置圆角的水平和垂直半径都是一样的，不过前面介绍过，

`border-radius` 设置圆角时, 可以把圆角的水平和垂直半径值单独设置, 此时就需要使用以“/”来区别。“/”前面的表示圆角的水平半径, 而“/”后面的值表示圆角的垂直半径。一起来看一个简单的实例, 设置不规则圆角边框。

```
.border-radius {
    width: 350px;
    height: 100px;
    border: 10px solid orange;
    border-radius: 60px 40px 30px 20px / 30px 20px 10px 5px;
}
```

`border-radius` 设置水平/垂直两个半径参数时, 元素的每个角不是四分之一圆角, 得到的圆角效果是不规则的。元素左上角的是一个水平半径为 60px, 垂直半径为 30px 的不规则圆角; 右上角是一个水平半径为 40px, 垂直半径为 20px 的不规则圆角; 右下角是一个水平半径为 30px, 垂直半径为 10px 的不规则圆角; 左下角是一个水平半径为 20px, 垂直半径为 5px 的不规则圆角, 如图 3-26 所示。

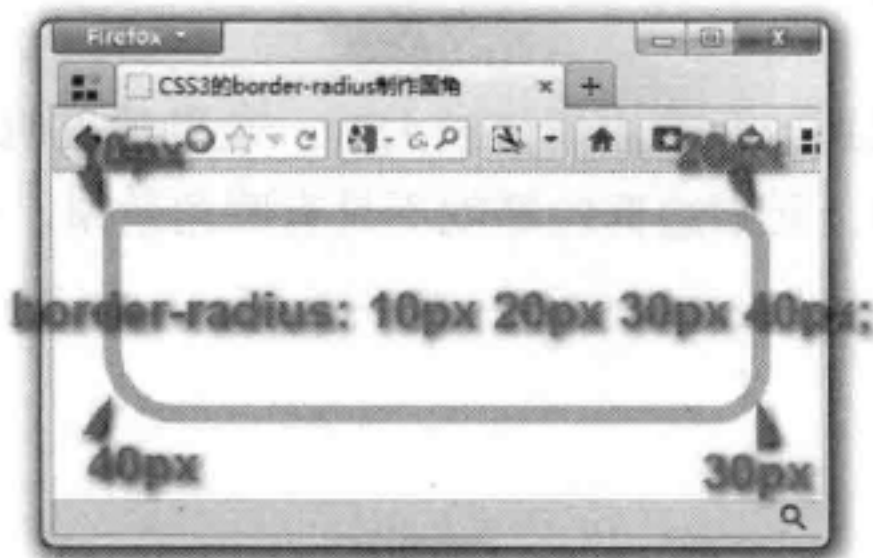


图 3-25 `border-radius` 设置四个值的圆角边框效果

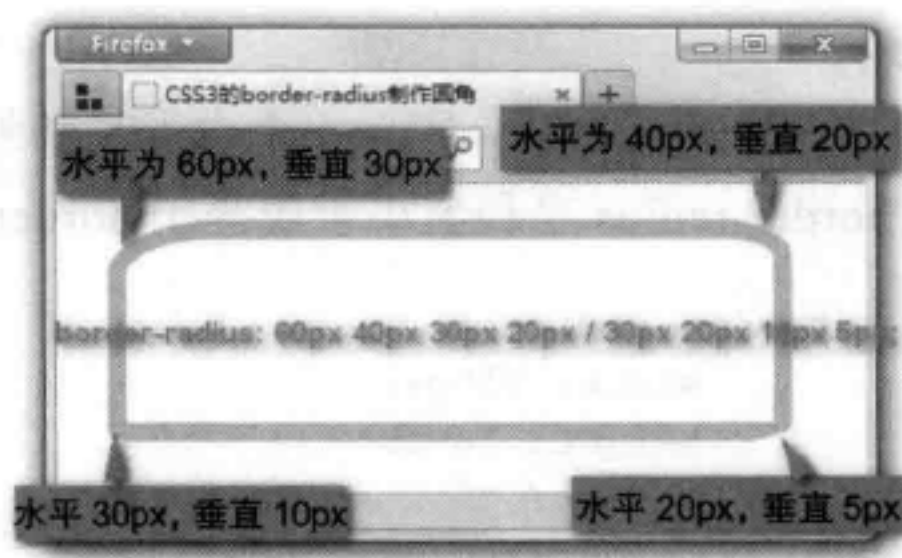


图 3-26 `border-radius` 设置不规则圆角边框效果

其实 `border-radius` 的水平和垂直半径也遵循前面介绍的规则, 可以设置 1~4 个值的集合, 同时它们分别遵循 CSS 赋值规则。但分开设置元素各个顶角的圆角的水平和垂直半径圆角效果时, 不需要“/”, 如下所示。

```
border-top-left-radius: 10px 50px;
border-top-right-radius: 20px 60px;
border-bottom-left-radius: 20px 60px;
border-bottom-right-radius: 30px 50px;
```

如果加上反而是一种错误的写法。

```
border-top-left-radius: 10px / 50px;
border-top-right-radius: 20px / 60px;
border-bottom-left-radius: 20px / 60px;
border-bottom-right-radius: 30px / 50px;
```

3. 制作单个圆角边框

使用 `border-radius` 可以给元素设置圆角边框，还可以使用 `border-radius` 的派生子属性来定义元素的圆角边框效果。

要给元素设置单个圆角效果，不一定需要使用 `border-radius` 派生的子属性，完全可以使用下面的方法来替代。

```
border-radius: 50px 0 0 0;
```

上面的代码就是给元素设置了一个左上角圆角边框效果，其效果等同于：

```
-moz-border-radius-topleft: 50px;
-webkit-border-top-left-radius: 50px;
border-top-left-radius: 50px;
```

这两种方法制作出来的效果都是一样的，如图 3-27 所示。

4. 特殊应用

前面所了解的都是 `border-radius` 一些常见的运用，其实 `border-radius` 还有几个特殊的应用。

1) `border-radius` 还有一个内半径和外半径的区别，元素边框值较大时，效果就很明显。当 `border-radius` 半径值小于或等于 `border` 的厚度时，元素边框内部就不具有圆角效果。如：

```
.border-radius {
    width: 350px;
    height: 100px;
    border: 30px solid orange;
    border-radius: 30px;
}
```

效果如图 3-28 所示。

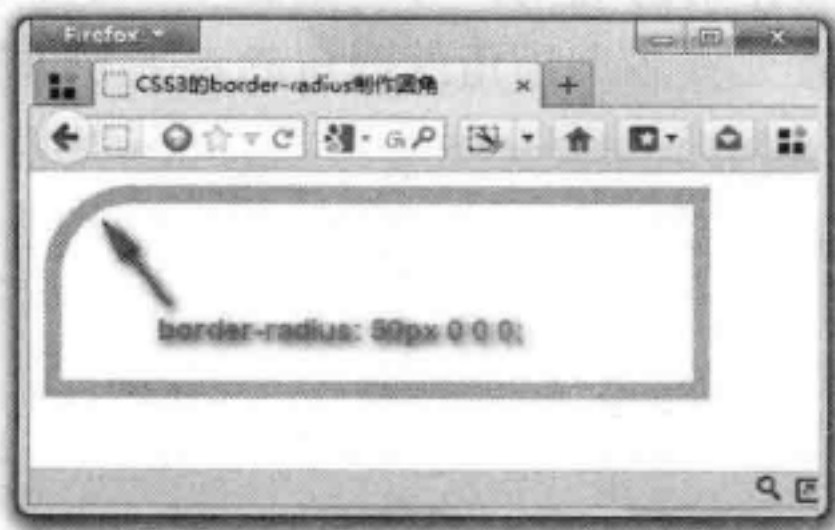


图 3-27 `border-radius` 制作单个圆角边框效果

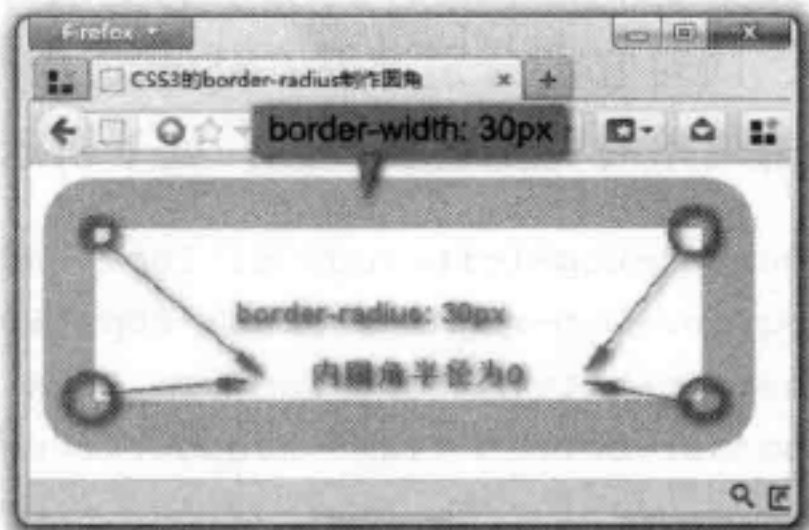


图 3-28 圆角半径等于边框厚度时效果

在这个基础上，把圆角半径值调大些，比边框值大。

```
.border-radius {
    width: 350px;
```



```

height: 100px;
border: 30px solid orange;
border-radius: 35px;
}

```

这个时候内圆角就出来了，如图 3-28 所示。

为何当 border-radius 的半径小于或等于元素的边框厚度时，内部是直角效果？因为 border-radius 内边半径（内径）等于外边半径（外径）减去对应的边框宽度。

□ border-radius 半径值与 border-width 值等于或小于 0 时，元素内角为直角，如图 3-28 所示。

□ border-radius 半径值与 border-width 值大于 0 时，元素内角具有圆角效果，其圆角半径为它们的差值。差值越大，圆角幅度也大，反之圆角幅度也小，如图 3-29 所示。

2) 第二种特殊应用是，元素相邻边有不同的宽度，这个角将会从宽的边平滑过渡到窄的一边，其中一条边甚至可以是 0，元素相邻转角是由大向小转。例如：

```

.border-radius {
width: 350px;
height: 100px;
border: 30px solid orange;
border-width: 20px 5px 30px 60px;
border-radius: 100px;
}

```

效果如图 3-30 所示。

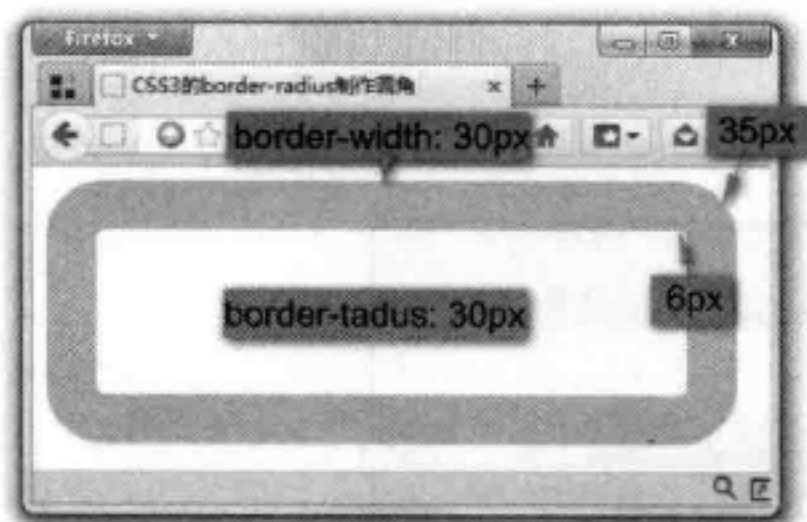


图 3-29 圆角半径大于边框厚度时效果

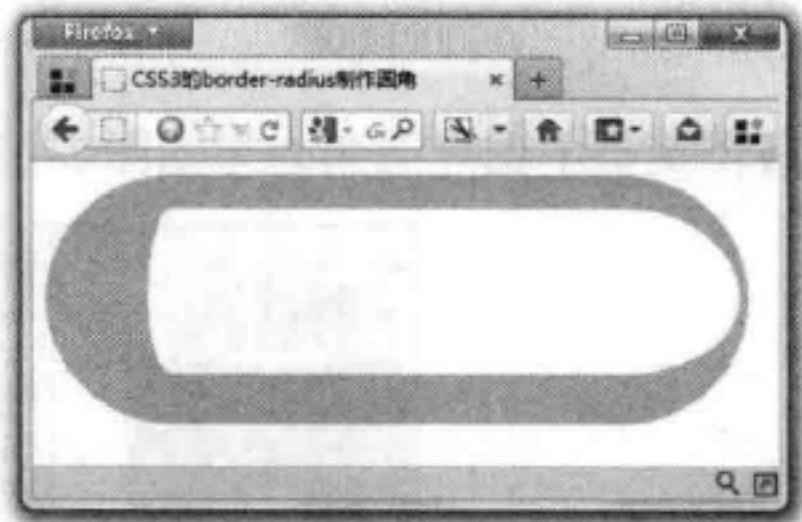


图 3-30 元素相邻转角是由大向小转效果

当元素相邻两条边颜色和线条样式不同时，两条相邻边颜色和样式转变的中心点是在一个和两边宽度成正比的角上。如果两条边宽度相同，这个临界点应该在一个 45 度角上，如果一条边是另外一条边的 2 倍，这个临界点就在一个 30 度角上。界定这个转变的线就是连接在内外曲线上的两个点的直线。一起来看个示例，给元素四边设置不同的颜色和宽度。

```

.border-radius {
width: 350px;
height: 100px;
}

```

```

border: 30px solid orange;
border-width: 35px 35px 60px 30px;
border-color: orange red green blue;
border-radius: 80px;
}

```

效果如图 3-31 所示。

5. 图片应用圆角

`border-radius` 能应用在各个元素中,但在 `img` 和 `table` 应用时会有点差别的,首先看图片上应用 `border-radius` 时的情况。在 `img` 上应有用 `border-radius` 到目前只有在 Webkit 内核浏览器不能对图片进行剪切,来看一个图片应用圆角的实例。

```

<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title> 图片上的圆角应用 </title>
  <style type="text/css">
    img {
      border: 5px solid red;
      border-radius: 10px;
    }
  </style>
</head>
<body>
  
</body>
</html>

```

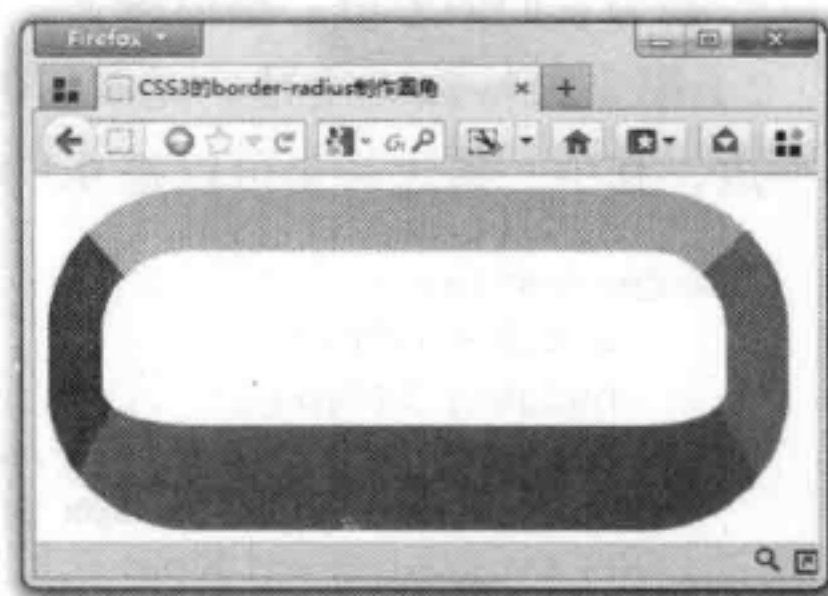


图 3-31 `border-radius` 临界分界点效果

各浏览器下图片圆角效果如图 3-32 所示。



图 3-32 各浏览器下图片圆角效果

正如图 3-32 效果所示, 图片在 Webkit 内核浏览器 (例如 Chrome、Safari) 下根本没有圆角效果, 图片不会被圆角剪切。如果需要对浏览器达到一致效果, 可以把图片转换成元素的背景图片, 然后再给元素定义圆角效果。这时需要借助 jQuery 来实现。例如:

```
$( "img" ).load(function() {
$(this).wrap(function() {
return '<span class="' + $(this).attr('class') + '" style="background:url(' +
$(this).attr('src') + ') no-repeat center center; width: ' +
$(this).width() + 'px; height: ' + $(this).height() + 'px;"></span>';
});
(this).css("opacity","0");
});
```

详细的解决方案可以参考 <http://www.w3cplus.com/css3/jquery-css3-rounded-image>。

6. 表格应用圆角

另外表格元素 table 使用 border-radius 是不一样的, 当表格样式属性 border-collapse 是 collapse 时, 表格不能正常显示, 只有 border-collapse 属性值为 separate 时, 表格圆角才能正常显示。例如:

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
<meta charset="UTF-8">
<title> 表格的圆角应用 </title>
<style type="text/css">
table {
margin: 10px;
border: 5px solid orange;
border-radius: 10px;
}
.table1 {
border-collapse: collapse;
}
.table2 {
border-collapse: separate;
}
</style>
</head>
<body>
<table class="table1">
<tr>
<td>border-collapse: collapse</td>
</tr>
</table>
<table class="table2">
<tr>
<td>border-collapse: separate</td>
</tr>
```

```

</table>
</body>
</html>

```

效果如图 3-33 所示。








图 3-33 各浏览器下表格圆角边框效果

3.4.3 浏览器兼容性

目前, border-radius 属性除了 IE 老版本之外的浏览器都得到了较好的支持, 如 IE 9+、Firefox 4+、Chrome 5+、Safari 5+、Opera 10.5+ 版本都支持 border-radius 的标准写法。如果需要支持一些老版本, 还要添加各浏览器的私有前缀, 如 Firefox 3、Chrome 4、Safari 3.1 ~ 4。而 IE 8 及其以下版本的浏览器不支持 border-radius 属性, 如表 3-6 所示。

表 3-6 border-radius 浏览器兼容性

属性名					
border-radius	9 + √	3.0 + √	1.0 + √	10.5 + √	3.0 + √

CSS3 的 border-radius 属性目前在 Web 中的使用随处可见, 特别是国外的 Web 运用上, 国内很多 Web 设计师也逐渐在使用。在 IE 低版本浏览器下, Web 设计师可以采用以下方案来处理兼容性。

- ❑ 使用第三方插件, 例如 IE -css3.js、PIE 或者其他 JavaScript 脚本插件。
- ❑ 采用渐近增强, 在不支持 border-radius 属性的浏览器采用另一套样式, 也就是 CSS2 中的图片实现圆角方法优雅降级, 在不支持 border-radius 的浏览器默认显示直角。

3.4.4 border-radius 属性的优势

使用 CSS 来实现宽度固定的圆角效果，采用背景图片配合滑动门技术实现还是一种不错的方法。但是，如果想要一个宽度不固定的元素就变得复杂了。宽度不定，就意味着这个元素在水平和垂直方向都能灵活地变化。实现元素四个圆角效果，就需要制作四个圆角背景图片，并且进行合理的放置，这样还需要添加四个额外的标签来辅助完成。当然还可以制作两个超大、超宽的背景图片，这个方法虽然减少了两张背景图片，以及四个 HTML 标签，但另一个问题又随之产生，图片尺寸过大增加了图片载入的难度，直接影响了网站的性能。如果使用 CSS3 的 border-radius 属性制作圆角，就不需考虑元素是否可以自由扩展，同时也不需要为了实现圆角制作不同的圆角背景图片，从而获得了极大的灵活性、维护性。

使用 CSS3 的 border-radius 属性来代替 CSS 之前使用图片制作圆角，在部分不支持 border-radius 的浏览器上，牺牲了一点效果的一致性，但这不是问题。我们所做的是一种渐进增强，一种优雅降级，即使用圆角的元素在不支持 CSS3 的 border-radius 属性的浏览器下完全有效果且易读，只不过在支持的浏览器下，使用 border-radius 的元素会更美观，视觉效果更细腻圆润。你或者客户希望在所有浏览器下能达到一样的圆角效果，可以考虑这样的实现方法：“在支持的浏览器下使用 CSS3 的 border-radius 属性，而在不支持的浏览器下，可以考虑图片，或者第三方插件的方法来实现”。

3.4.5 实战体验：制作特殊图形

border-radius 属性除了可以实现元素的圆角效果，还可以制作一些特殊的图形效果，如圆形、半圆形、扇形、椭圆形和不规则的圆角图形等。

1. 圆形

border-radius 制作圆角有两点技巧。

❑ 元素的宽度和高度相同。

❑ 圆角的半径值为元素宽度或高度的一半或者直接设置圆角半径值为 50%。

不过早期的 Webkit 内核浏览器不支持百分比值。例如：

```
div {  
    width: 100px;  
    height: 100px;  
    background-color: orange;  
    border-radius: 50%;  
}
```

效果如图 3-34 所示。

2. 半圆

border-radius 制作半圆与制作圆形的方法是一样的，只是元素的宽度与圆角方位要配

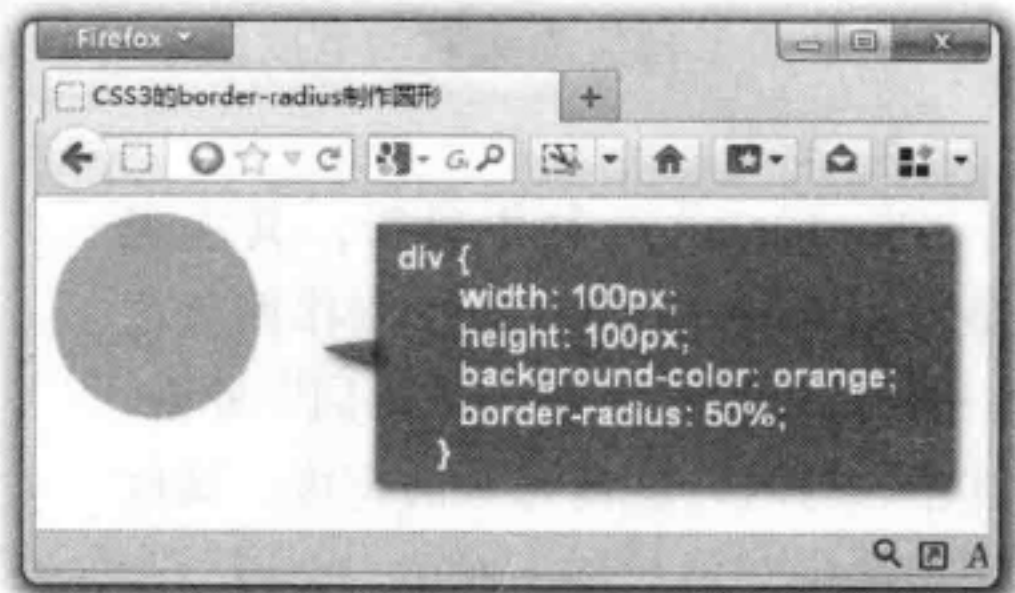


图 3-34 border-radius 制作圆形

合一致，不同的宽度和高度比例，以及圆角方位，可以制作上半圆、下半圆、左半圆和右半圆效果。例如：

```
.semicircle {
    background-color: orange;
    margin: 30px;
}
.top {
    width: 100px; /* 宽度为高度的 2 倍 */
    height: 50px;
    border-radius: 50px 50px 0 0; /* 圆角半径为高度的值 */
}
.right {
    height: 100px; /* 高度为宽度的 2 倍 */
    width: 50px;
    border-radius: 0 50px 50px 0; /* 圆角半径为宽度的值 */
}
.bottom {
    width: 100px; /* 宽度为高度的 2 倍 */
    height: 50px;
    border-radius: 0 0 50px 50px; /* 圆角半径为高度的值 */
}
.left {
    width: 50px;
    height: 100px; /* 高度为宽度的 2 倍 */
    border-radius: 50px 0 0 50px; /* 圆角半径为宽度的值 */
}
```

效果如图 3-35 所示。

border-radius 制作半圆有两个小技巧：

- ❑ 制作上半圆或下半圆，元素的宽度值是高度值的 2 倍，而且圆角半径值为元素的高度值；
- ❑ 制作左半圆或右半圆，元素的高度值是宽度值的 2 倍，而且圆角半径值为元素的宽度值。

3. 扇形

border-radius 制作扇形，其实就是使用 **border-radius** 属性制作四分之一圆形。遵循“三同，一不同”原则，其中“三同”是指元素的宽度、高度和圆角半径值相同，而“一不同”指的是圆角位置不同。根据圆角取值位

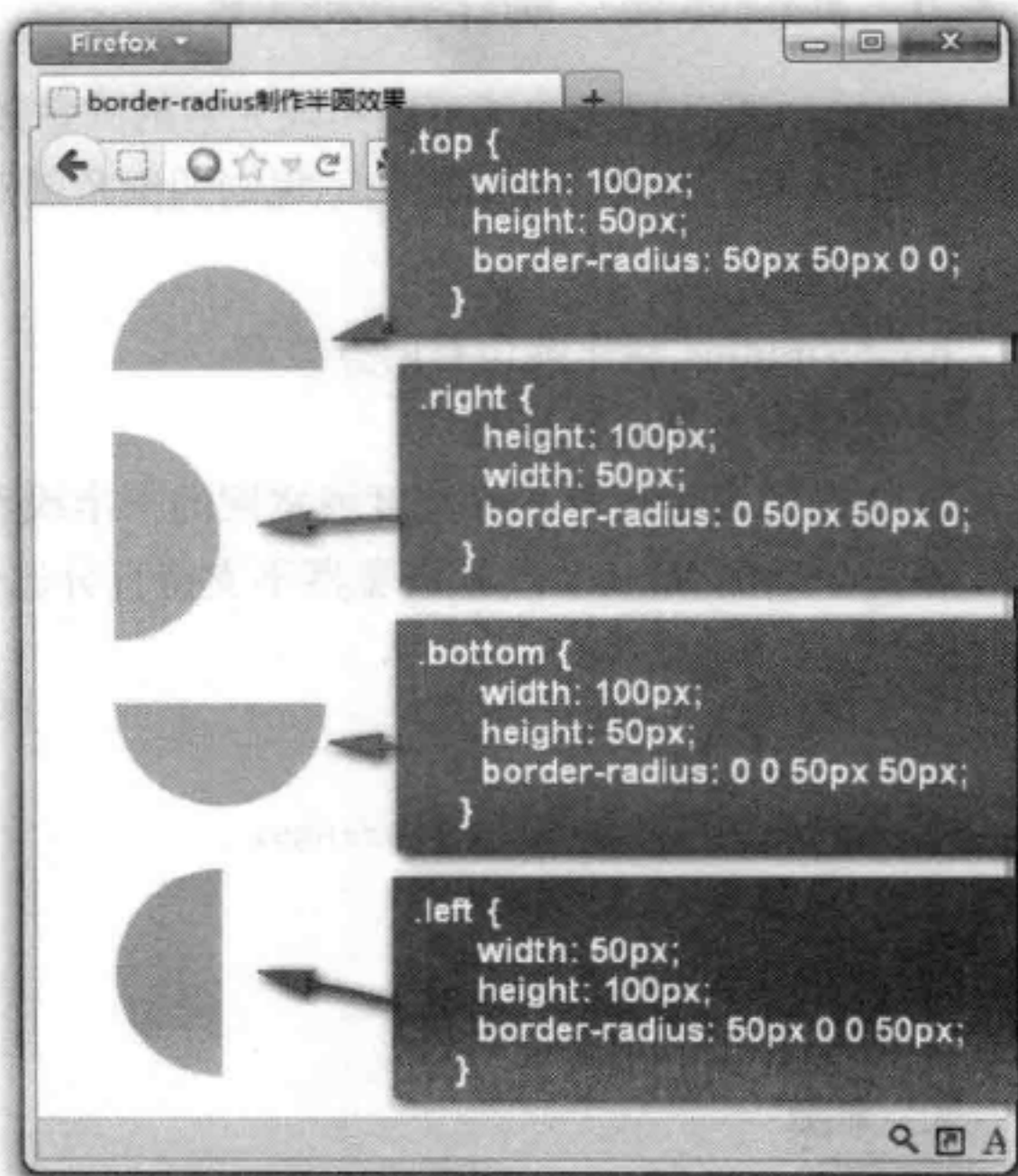


图 3-35 border-radius 制作半圆形

置不一样，可以分左上、右上、右下和左下四种扇形效果。例如：

```
.quarterCircle {
    background-color: orange;
    margin: 30px;
}
.top {
    width: 100px;
    height: 100px;
    border-radius: 100px 0 0 0;
}
.right {
    width: 100px;
    height: 100px;
    border-radius: 0 100px 0 0;
}
.bottom {
    width: 100px;
    height: 100px;
    border-radius: 0 0 100px 0;
}
.left {
    width: 100px;
    height: 100px;
    border-radius: 0 0 0 100px;
}
```

效果如图 3-36 所示。

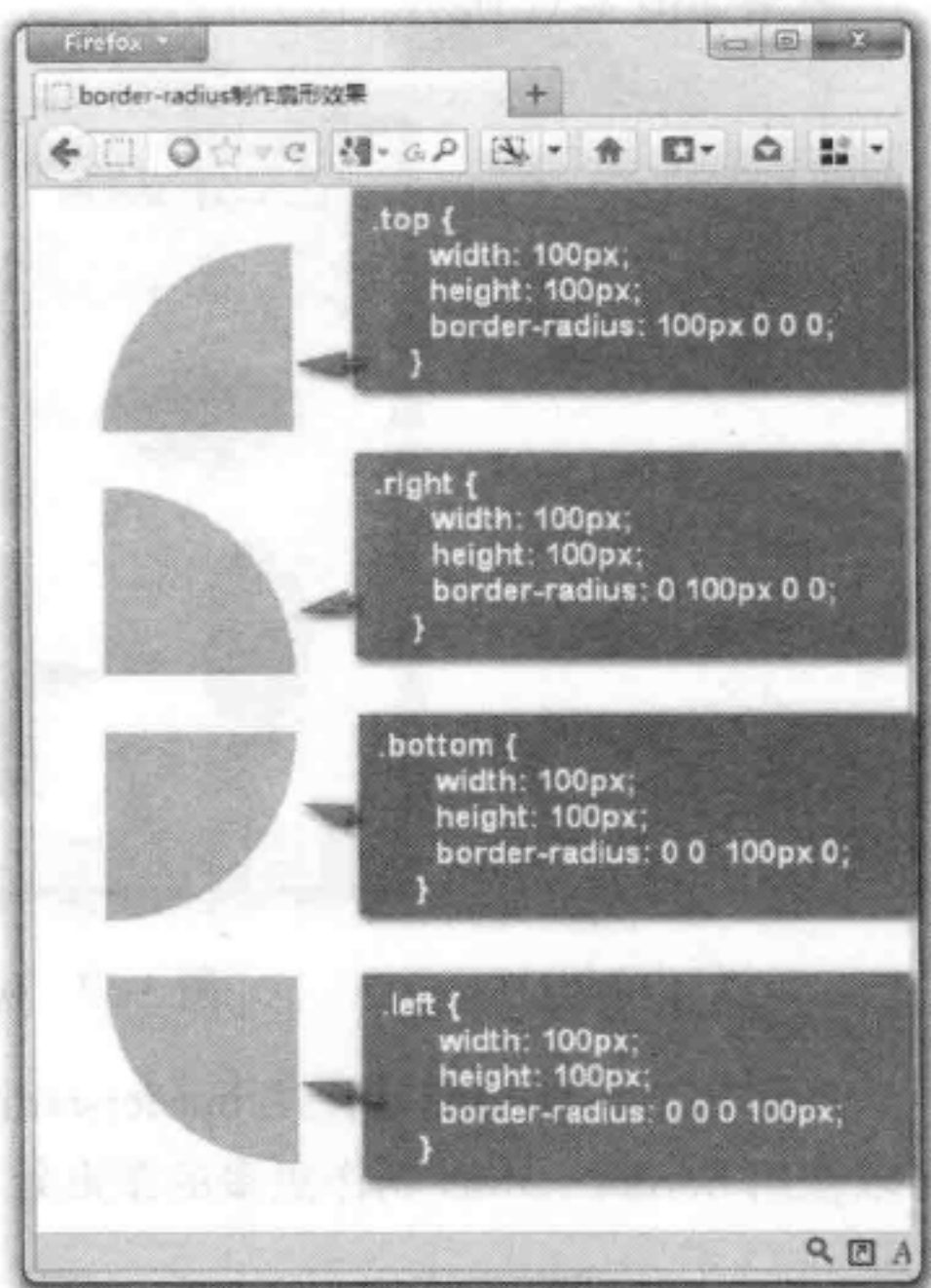


图 3-36 border-radius 制作四分之一圆

4. 椭圆

椭圆其实就是一个圆形受到挤压而成的一种形状，border-radius 制作椭圆也非常方便，只受限于元素的宽度或高度，然后就是圆角半径，制作椭圆的圆角半径和其他图形有所不同，需要设置圆角的水平和垂直方向的半径值。椭圆有两种，一种是水平的，另外一种垂直的。它们之间的差别只是方向性的区别，其制作方法是一样的。

制作水平椭圆，元素宽度是高度的 2 倍，而且 border-radius 的水平半径等于元素宽度，垂直半径等于元素高度；而垂直椭圆刚好与水平椭圆的参数相反。例如：

```
.oval {
    background-color: orange;
    margin: 30px;
}
.hov {
    width: 100px;
    height: 50px;
    border-radius: 100px / 50px;
}
.ver {
    width: 50px;
    height: 100px;
}
```

```
border-radius: 50px / 100px;
}
```

效果如图 3-37 所示。

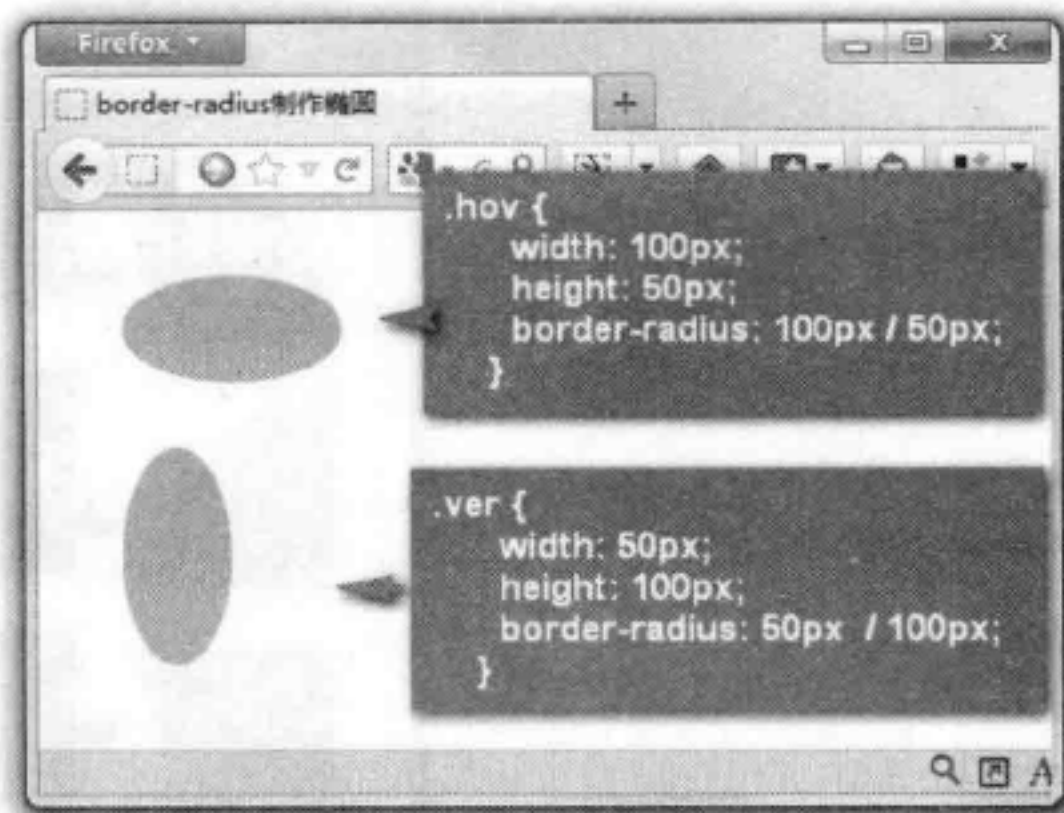


图 3-37 border-radius 制作椭圆

上面几个小案例都是使用 border-radius 配合元素的其他属性实现不同的图形效果，也可以使用 border-radius 制作更多的图形效果（或者说圆角效果）来适合项目需求。

3.5 CSS3 盒子阴影属性

box-shadow 也是 CSS3 新增的一个重要属性，用来定义元素的盒子阴影。本节主要介绍 CSS3 的 box-shadow 的属性以及如何使用。

3.5.1 box-shadow 属性的语法及参数

在具体学习 box-shadow 使用方法之前，我们必须先知道 box-shadow 使用的语法规则。

```
box-shadow: none | [ <length> <length> <length>? <length>? || <color> ] [ , <length> <length> <length>? <length>? || <color> ] +
```

上面的语法规则可以简写如下：

```
box-shadow: none | [inset x-offset y-offset blur-radius spread-radius color], [inset x-offset y-offset blur-radius spread-radius color]
```

box-shadow 属性可以使用一个或多个投影，如果使用多个投影时必须使用逗号“,”隔开。

其实 box-shadow 属性很简单，可以为其设置以下参数。

- ❑ none: 默认值, 元素没有任何阴影效果。
- ❑ inset: 阴影类型, 可选值。如果不设置, 其默认的投影方式是外阴影; 如果取其唯一值 “inset”, 就是给元素设置内阴影。
- ❑ x-offset: 阴影水平偏移量, 其值可以是正负值。如果取正值, 则阴影在元素的右边, 反之取负值, 阴影在元素的左边。
- ❑ y-offset: 阴影垂直偏移量, 其值可以是正负值。如果取正值, 则阴影在元素的底部, 反之取负值, 阴影在元素的顶部。
- ❑ blur-radius: 阴影模糊半径, 可选参数。其值只能是正值, 如果取值为 “0” 时, 表示阴影不具有模糊效果, 如果取值越大, 阴影的边缘就越模糊。
- ❑ spread-radius: 阴影扩展半径, 可选参数。其值可以是正负值, 如果取值为正值, 则整个阴影都延展扩大, 反之取值为负值, 则整个阴影都缩小。
- ❑ color: 阴影颜色, 可选参数, 如果不设定任何颜色时, 浏览器会取默认色, 但各浏览器默认色不一样, 特别是在 Webkit 内核下的浏览器将无色, 也就是透明, 建议不要省略这个参数。

3.5.2 box-shadow 属性使用方法

和 PSD 软件制作图片相比, box-shadow 修改元素的阴影效果要方便得多, 因为 box-shadow 可以修改六个参数, 得到不同的效果。下面结合一些简单的案例来对 box-shadow 属性进行演示说明。

1. 单边阴影效果

定义元素的单边阴影效果和调协 border 的单边边框颜色是相似的, 例如:

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>box-shadow 设置单边阴影效果 </title>
  <style type="text/css">
    .box-shadow {
      width: 200px;
      height: 100px;
      border-radius: 5px;
      border: 1px solid #ccc;
      margin: 20px;
    }
    .top {
      box-shadow: 0 -2px 0 red;
    }
    .right {
      box-shadow: 2px 0 0 green;
    }
  </style>
</head>
</html>
```

```

    .bottom {
        box-shadow: 0 2px 0 blue;
    }
    .left {
        box-shadow: -2px 0 0 orange;
    }
</style>
</head>
<body>
    <div class="box-shadow top"></div>
    <div class="box-shadow right"></div>
    <div class="box-shadow bottom"></div>
    <div class="box-shadow left"></div>
</body>
</html>

```

效果如图 3-38 所示。

这个案例中，使用 box-shadow 给元素设置了顶边、右边、底边和左边的单边阴影效果。主要通过 box-shadow 的水平和垂直阴影的偏移量来实现，其中 x-offset 为正值时，生成右边阴影，反之为负值时，生成左边阴影；y-offset 为正值时，生成底部阴影，反之为负值时生成顶部阴影。此例中是一个单边实影投影效果（阴影模糊半径为 0），但是如果阴影的模糊半径不是 0，上面的方法还能不能实现单边阴影效果呢？不急着来回答，在上面的实例中添加一个模糊半径，例如：

```

.top {
    box-shadow: 0 -2px 5px red;
}
.right {
    box-shadow: 2px 0 5px green;
}
.bottom {
    box-shadow: 0 2px 5px blue;
}
.left {
    box-shadow: -2px 0 5px orange;
}

```

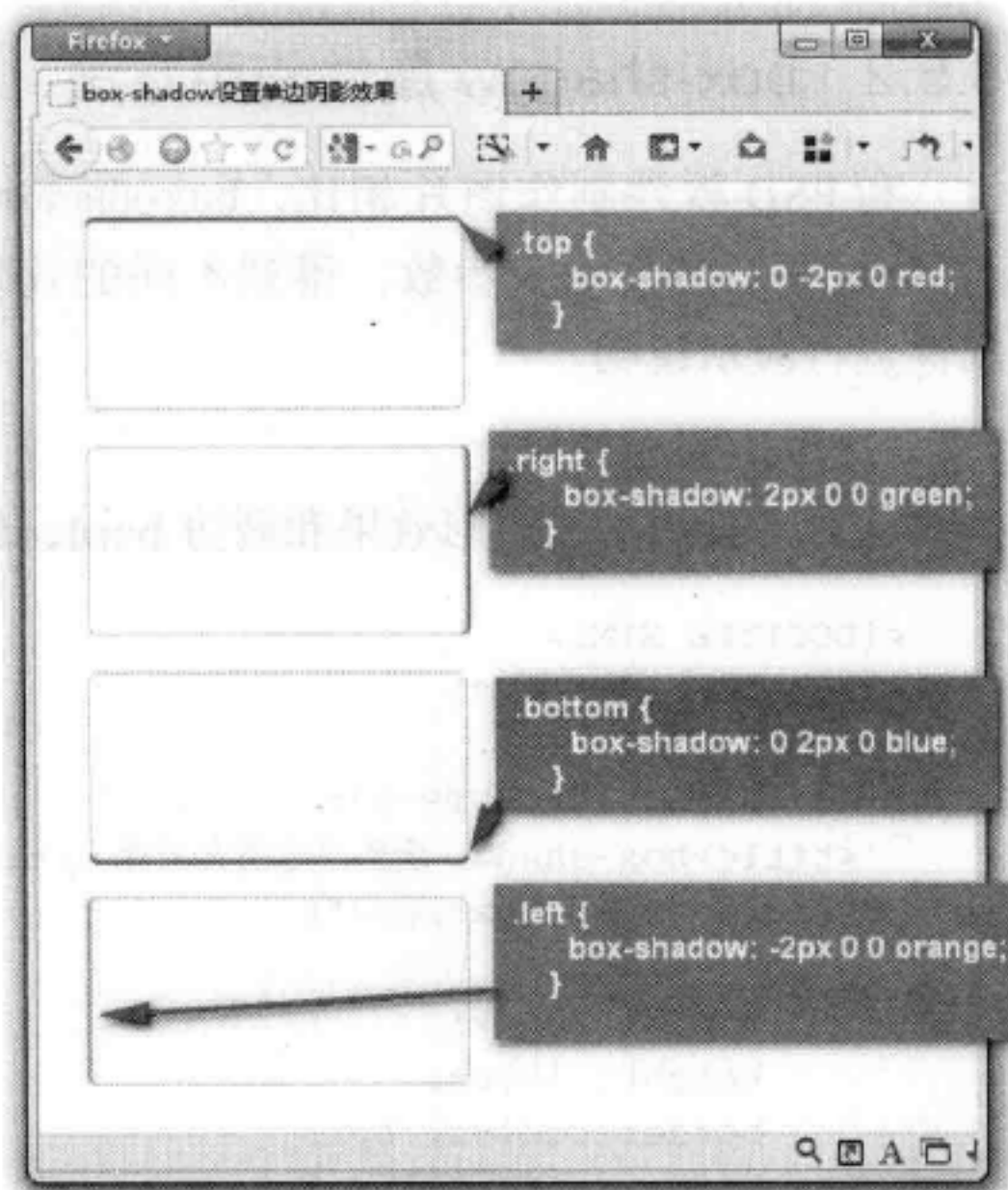


图 3-38 box-shadow 单边阴影效果

图 3-39 说明，这个效果并不是理想的单边阴影效果，当 box-shadow 添加了 5px 阴影模糊半径后，阴影不再是实影投影，阴影清晰度向外扩散，更具阴影的效果。但造成了另一个问题，给元素其他三个边加上淡淡的阴影效果，可这并不是设计需要的效果。

那究竟要怎么做呢？此时，box-shadow 属性中的阴影扩展半径（spread-radius）会是一

个很关键的属性，要实现单边阴影效果，必须配上这个属性（除单边实影之外）。

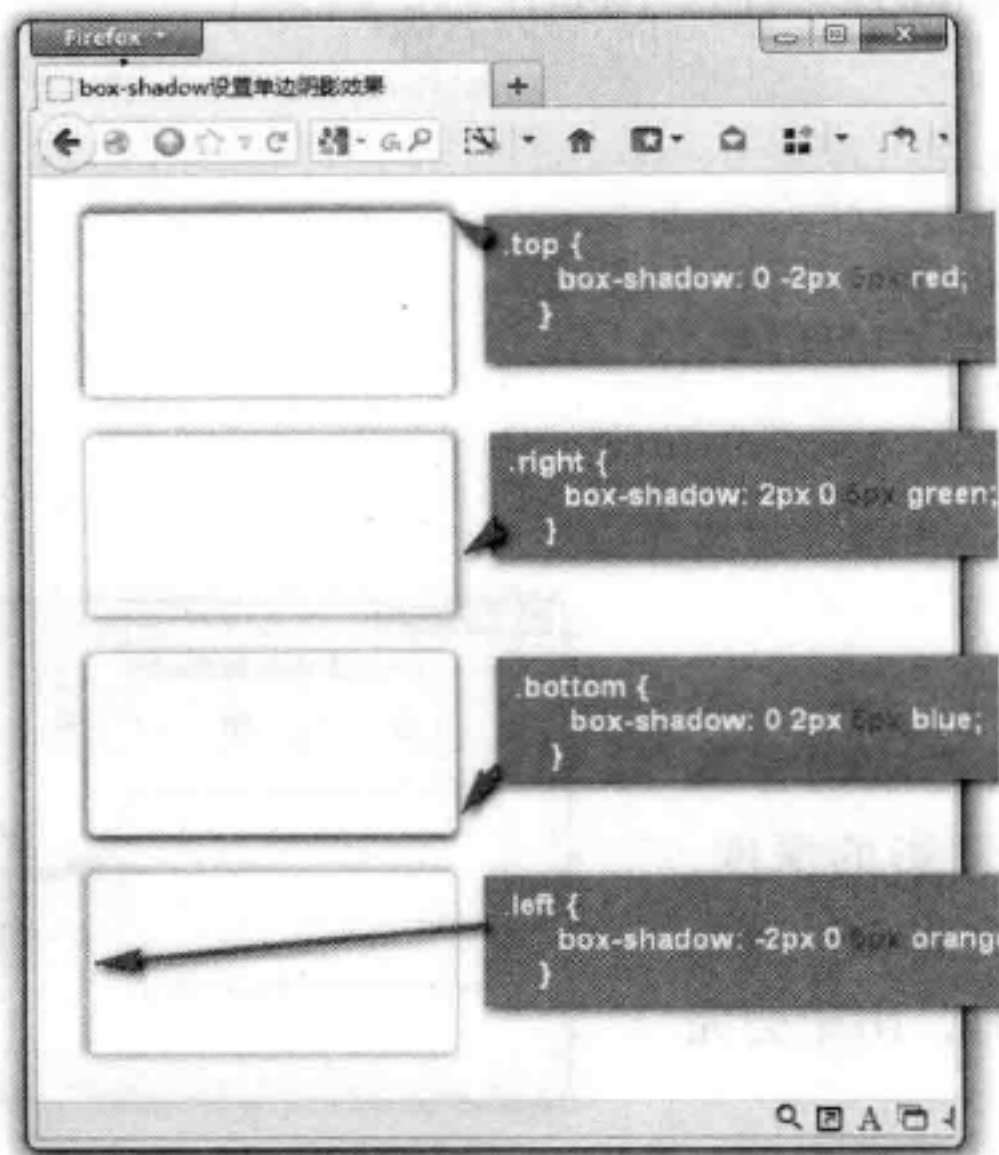


图 3-39 有模糊值的单边阴影效果

```
.top {
    box-shadow: 0 -4px 5px -3px red;
}
.right {
    box-shadow: 4px 0 5px -3px green;
}
.bottom {
    box-shadow: 0 4px 5px -3px blue;
}
.left {
    box-shadow: -4px 0 5px -3px orange;
}
```

上面的代码调整了阴影的位移量，新增了 box-shadow 的扩展半径，最终效果如图 3-40 所示。

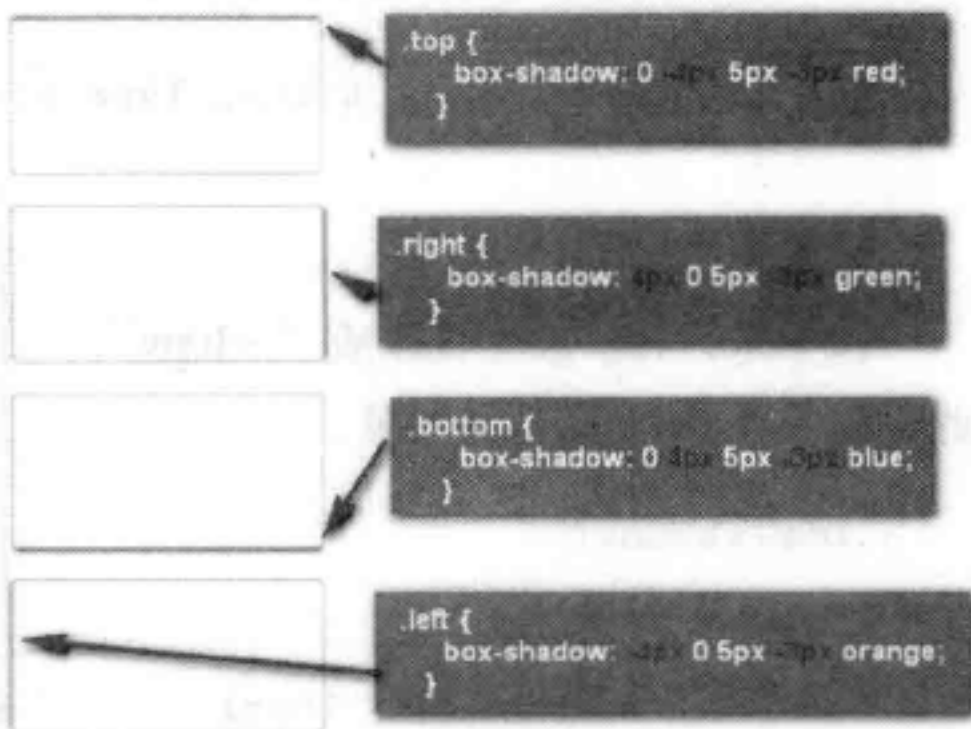


图 3-40 box-shadow 制作单边阴影效果



注意 各浏览器下显示效果略有细节差别。

2. 四边相同阴影效果

box-shadow 给元素设置相同的四边阴影效果，其实分为两种，在这里先看第一种。

(1) 只设置阴影模糊半径和阴影颜色

只设置阴影模糊半径和阴影颜色。例如：

```
.box-shadow{
    width: 200px;
    height: 100px;
    border-radius: 10px;
    border: 1px solid #ccc;
    margin: 20px;
    box-shadow: 0 0 10px #06c;
}
```

效果如图 3-41 所示。

在这个示例基础上，添加 box-shadow 扩展半径还可以控制阴影深度，如果取正值将加深阴影的深度，如果取负值可以向内压缩阴影，直到扩展半径等于模糊半径时，阴影会完全消失。例如：

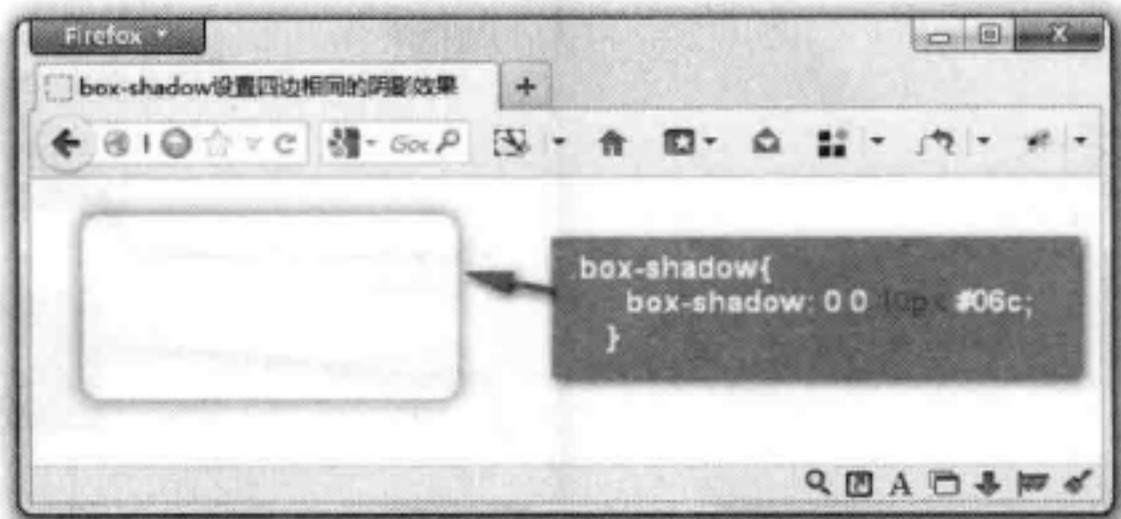


图 3-41 box-shadow 设置四边相同阴影

```
.box-shadow{
    width: 200px;
    height: 100px;
    border-radius: 10px;
    border: 1px solid #ccc;
    margin: 20px;
    box-shadow: 0 0 10px 10px #06c;
}
```

效果如图 3-42 所示。

接下来，将扩展半径改成“-10px”，此时将看不到任何阴影效果。

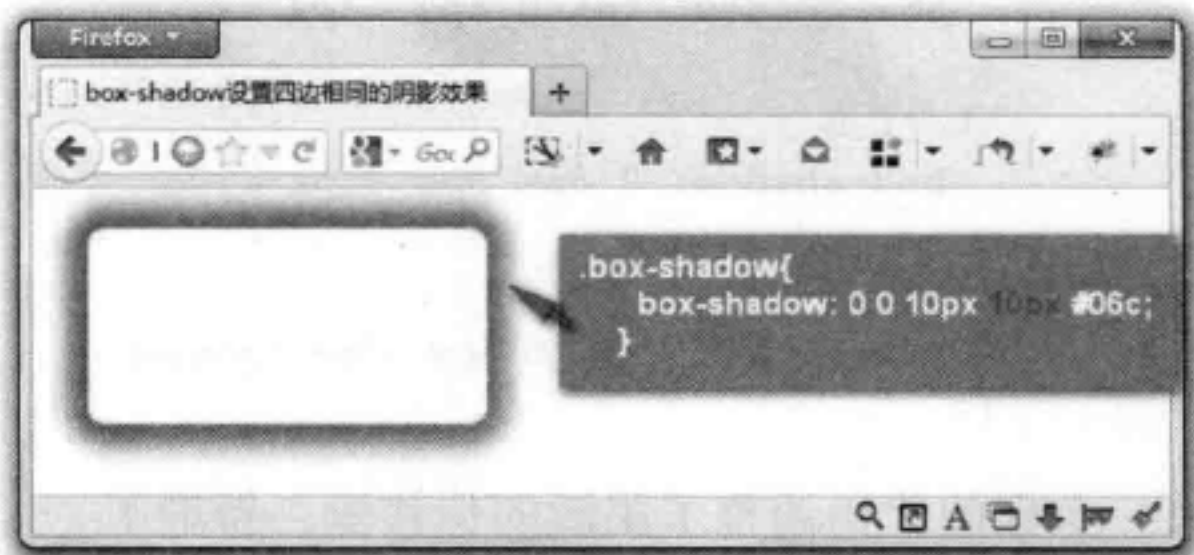


图 3-42 box-shadow 扩展半径加强阴影效果

```
.box-shadow{
    width: 200px;
    height: 100px;
    border-radius: 10px;
    border: 1px solid #ccc;
    margin: 20px;
    box-shadow: 0 0 10px -10px #06c;
}
```

效果如图 3-43 所示。

(2) 只设置扩展半径和阴影颜色

另外一种设置元素四边相同阴影效果，是设置扩展半径和阴影颜色，先来看一个简单的示例。



图 3-43 无阴影效果

```
.box-shadow{
    width: 200px;
    height: 100px;
    border-radius: 10px;
    border: 1px solid #ccc;
    margin: 20px;
    box-shadow: 0 0 0 10px #06c;
}
```

效果如图 3-44 所示。

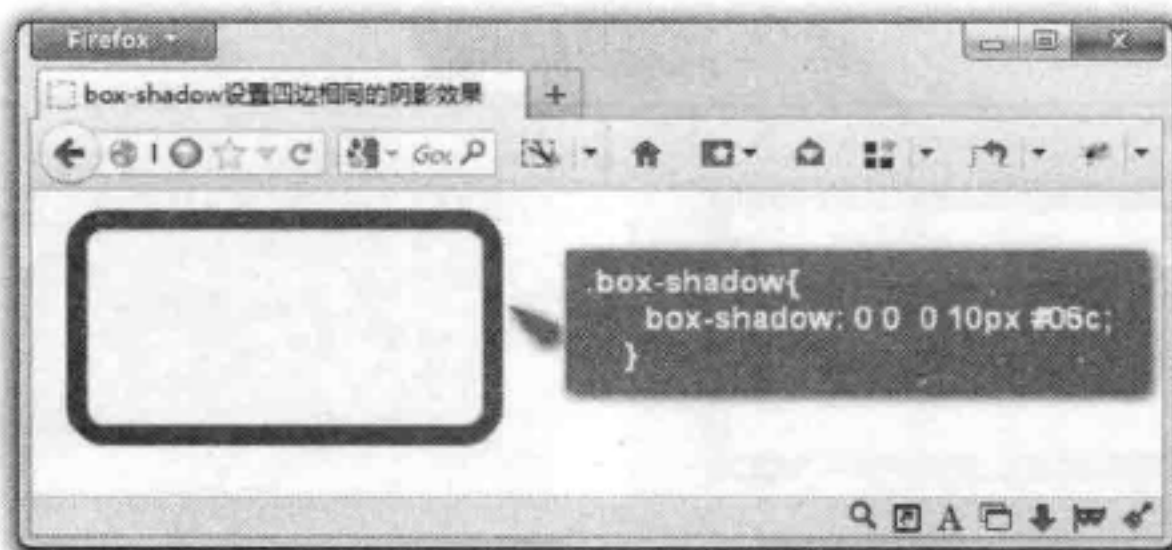


图 3-44 四边相同阴影效果

从图 3-44 可知道，box-shadow 制作的阴影效果和元素设置“10px”实线边框一样。

```
border:10px solid #06c;
```

如此一来，可以利用 box-shadow 扩展半径制作类似于边框的效果，但实质上并非边框，因为 box-shadow 并不是盒模型中的元素，不会计算到内容宽度。具体来看一个 box-shadow 与 border 的对比示例。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
    <meta charset="UTF-8">
    <title>border 与 box-shadow</title>
```

```

<style type="text/css">
  .box {
    width: 200px;
    height: 100px;
    text-align: center;
    line-height: 100px;
    float: left;
    margin: 30px;
  }
  .border{
    border: 10px solid red;
  }
  .box-shadow {
    box-shadow: 0 0 0 10px red;
  }
</style>
</head>
<body>
  <div class="box border">Border</div>
  <div class="box box-shadow">Box-shadow</div>
</body>
</html>

```

效果如图 3-45 所示。

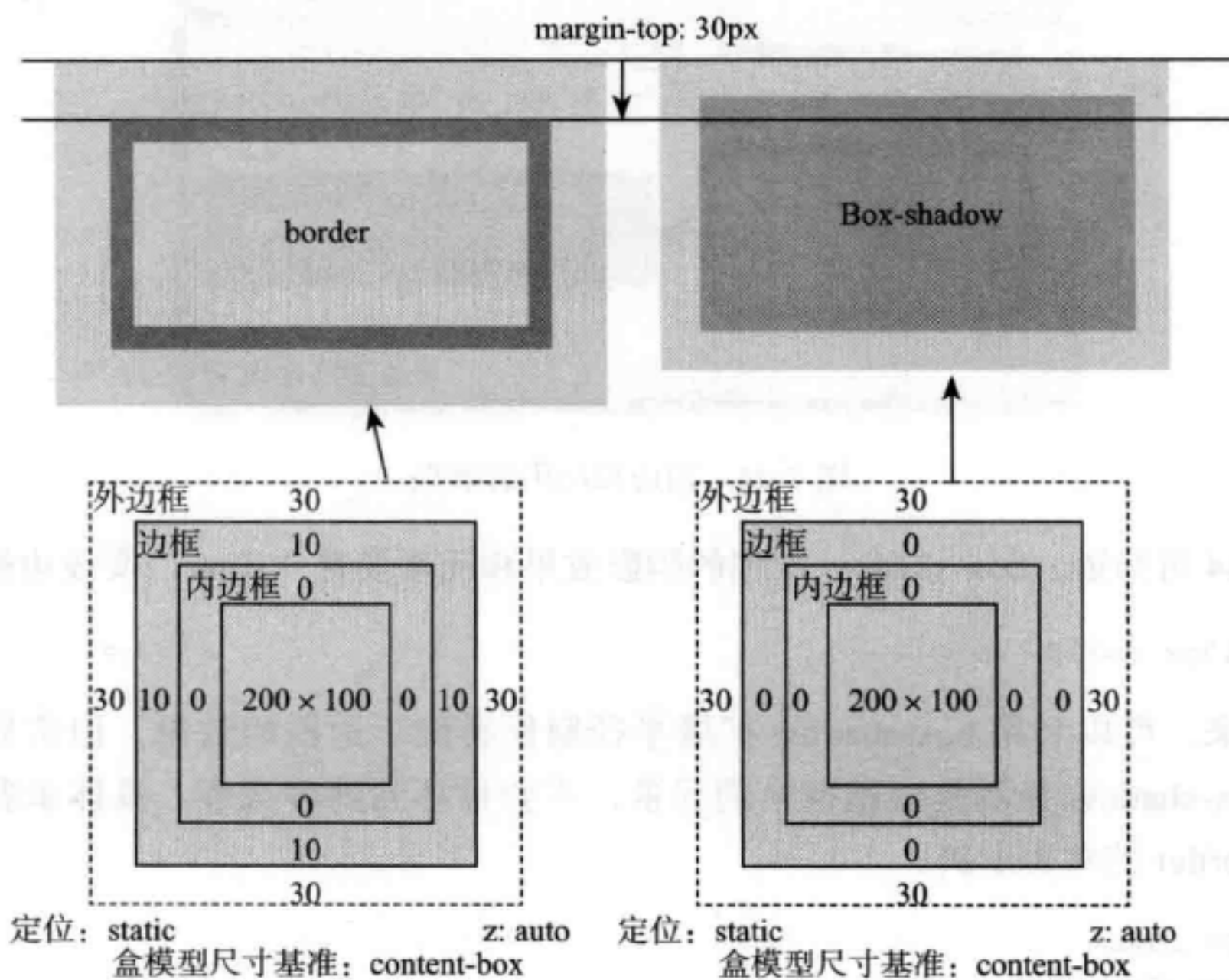


图 3-45 border 与 box-shadow 制作边框效果对比

图 3-45 证实了 box-shadow 不会影响页面的任何布局。div.border 元素的边框被计算了

宽度，但 `div.box-shadow` 的阴影被浏览器忽略不计，所以借助 `box-shadow` 属性的这个特性，`border-shadow` 用来模拟元素的边框效果可以自由地使用，但必须注意其层级关系。

W3C 标准规范中描述了 `box-shadow` 的工作方式，直观告诉我们 `box-shadow` 在元素盒模型中的层次关系，如图 3-46 所示。

图 3-46 告诉我们很多信息，比如说 `border-radius` 圆角、阴影扩展、阴影模糊以及 `padding` 是如何影响对象的阴影的。非零值的 `border-radius` 会以相同的作用影响阴影的外形，但 `border-image` 不会影响对象阴影的任何外形；对象阴影同盒模型的层次一样，外阴影会在对象背景之上，内阴影会在边框之下，背景之上。所以整个层级就是：边框在内阴影之上，内阴影在背景图片之上，背景图片在背景色之上，背景色在外阴影之上。

3. 内阴影

前几种都是外阴影的使用方法，其实使用 `inset` 属性值可以改变元素的阴影类型，将元素的默认外阴影重置为内阴影类型。例如：

```
.box-shadow {
    width: 200px;
    height: 100px;
    border: 1px solid #ccc;
    border-radius: 5px;
    box-shadow: inset 3px 3px 10px #06c;
}
```

效果如图 3-47 所示。

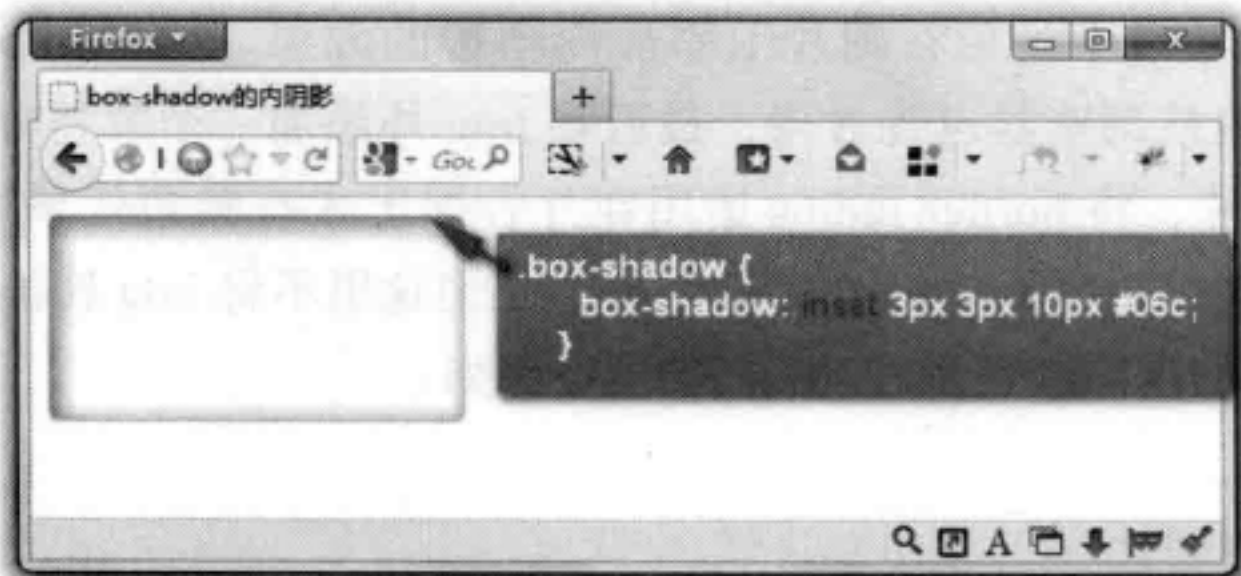


图 3-47 `box-shadow` 制作内阴影

不过 `box-shadow` 的内阴影使用在图片“`img`”元素上是没有任何效果的，例如：

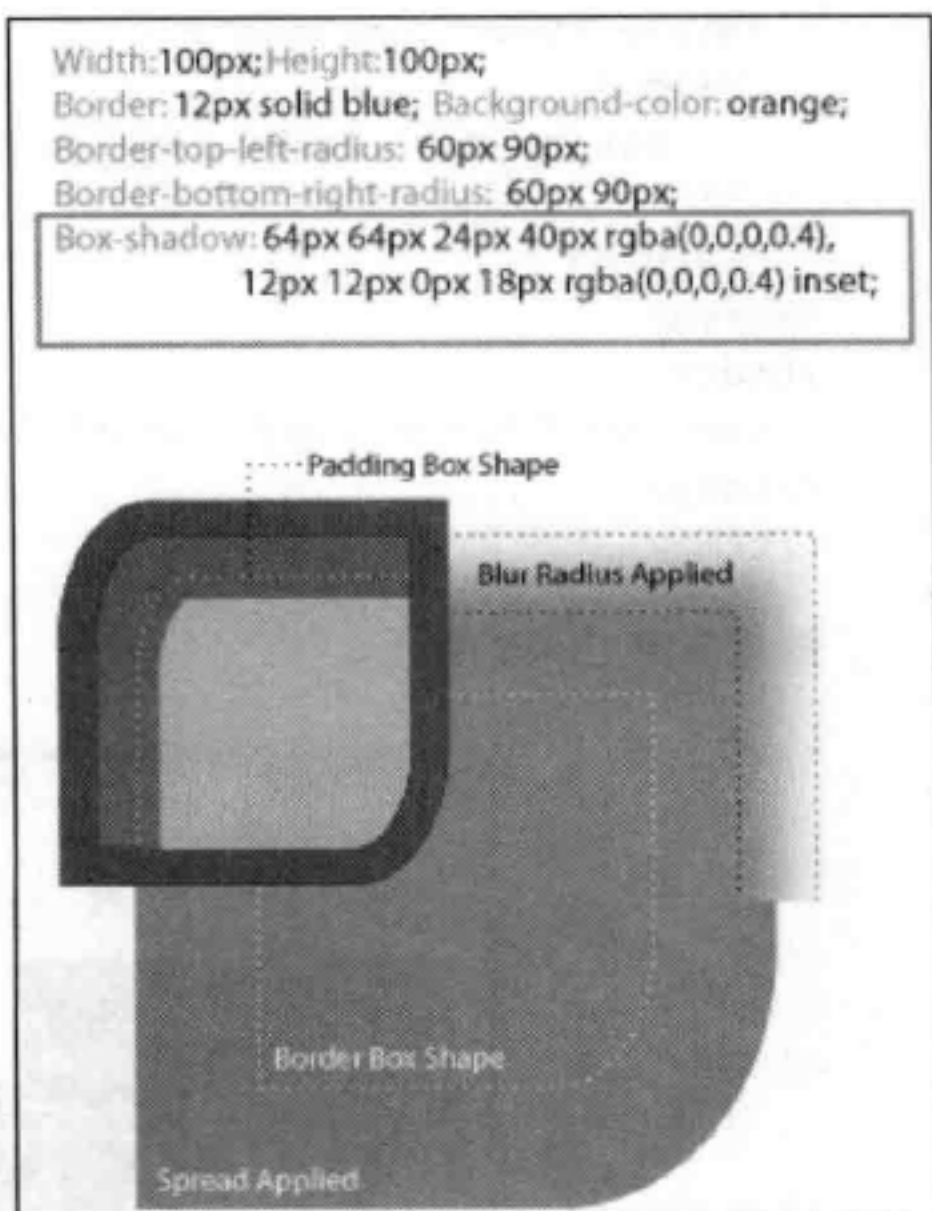


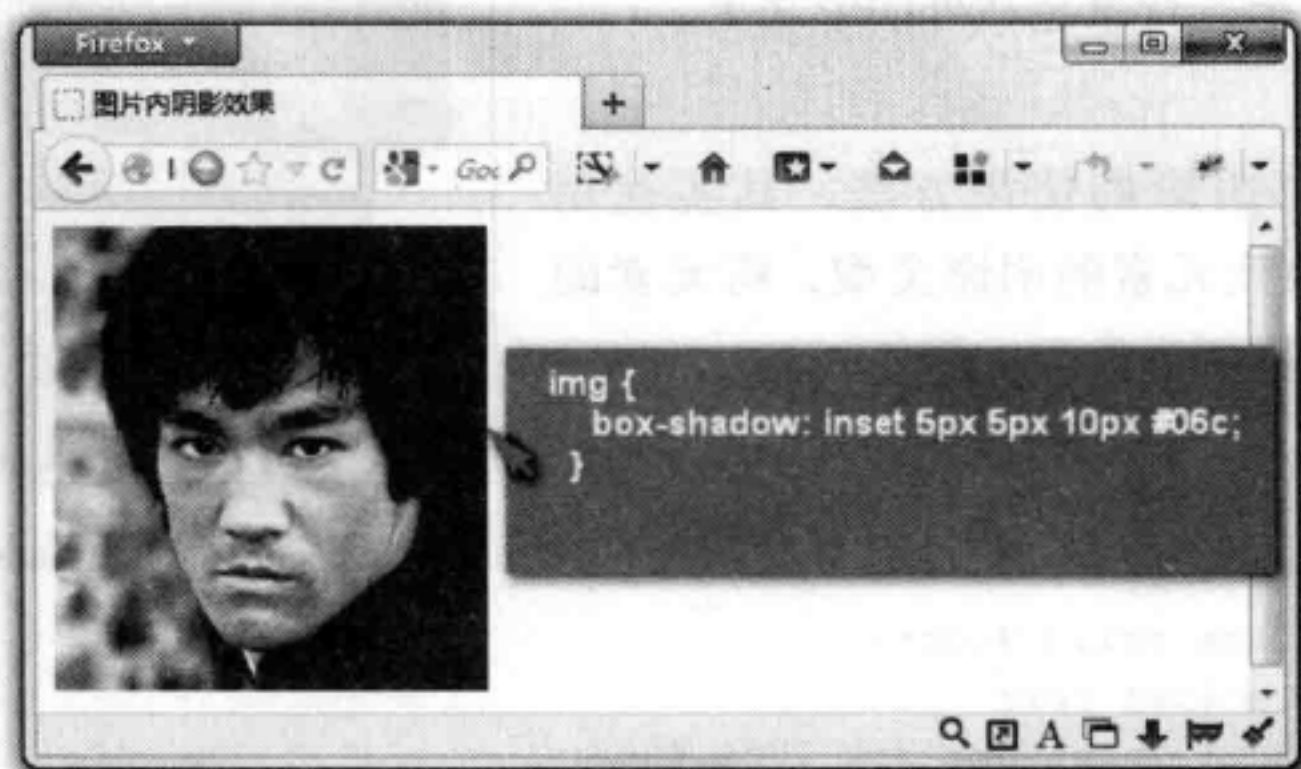
图 3-46 `box-shadow` 的工作方式

```

<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title> 图片内阴影效果 </title>
  <style type="text/css">
    img {
      box-shadow: inset 5px 5px 10px #06c;
    }
  </style>
</head>
<body>
  
</body>
</html>

```

效果如图 3-48 所示。



☆图 3-48 box-shadow 在 img 上的内阴影无效果

图 3-48 的效果再次证实了 box-shadow 的 inset 内阴影直接运用在 img 上没有任何效果，但在实际 Web 项目中难免在图片上添加内阴影的效果。记得将 border-radius 运用在 img 上时，Webkit 内核浏览器也无效果，最后在 img 外添加一个容器标签，并将 img 转换成外容器的背景图片，将 border-radius 运用在外容器上才有圆角效果的。借助这个思路，也在 img 标签外添加一个容器，例如“div”标签，但这里不将 img 转换成 div 标签的背景，只是将 box-shadow 的内阴影使用在 div 标签上，例如：

```

<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title> 图片内阴影效果 </title>
  <style type="text/css">
    .box-shadow {

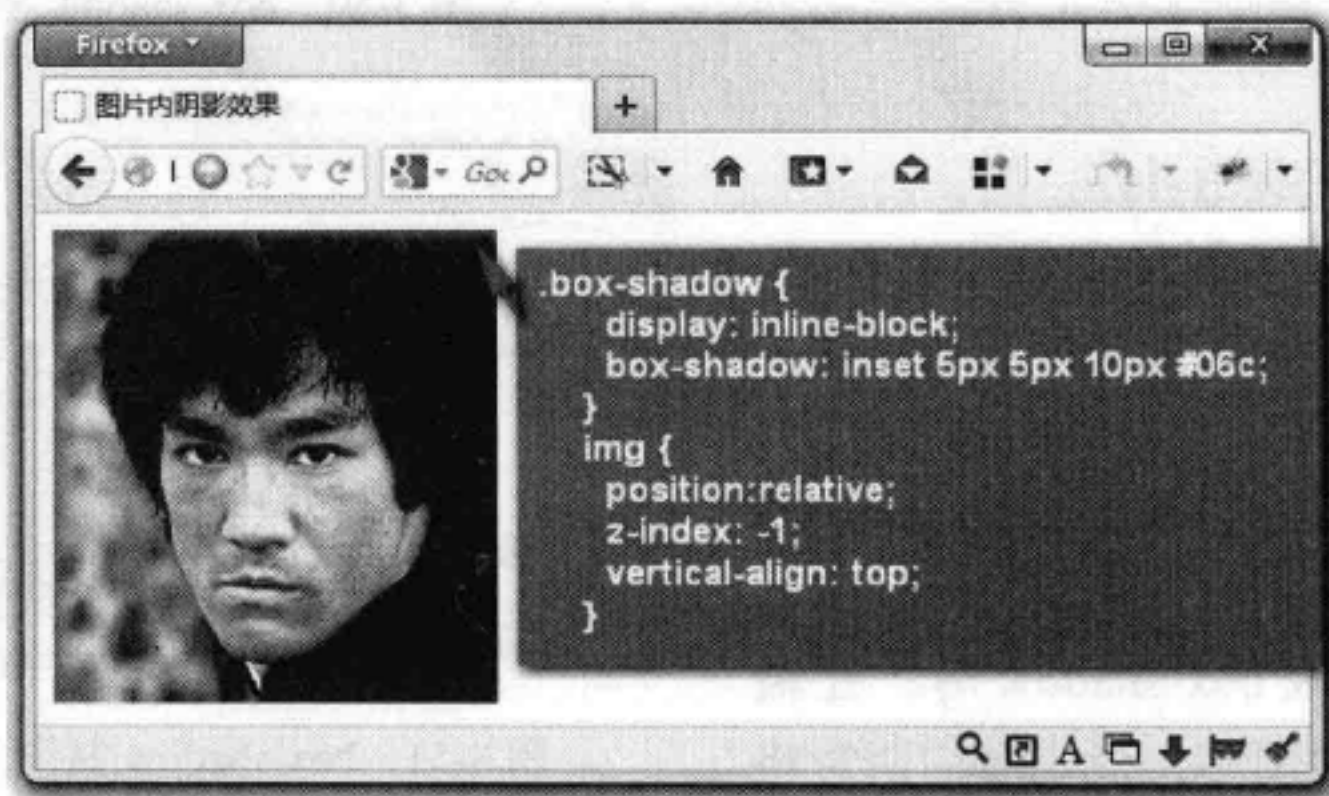
```

```

    display: inline-block; /* 这个很重要 */
    box-shadow: inset 5px 5px 10px #06c;
}
img {
    position: relative; /* 这个很重要 */
    z-index: -1; /* 这个很重要 */
    vertical-align: top;
}
</style>
</head>
<body>
<div class="box-shadow">
    
</div>
</body>
</html>

```

此时 img 就具有内阴影效果了, 如图 3-49 所示。



☆图 3-49 图片内阴影效果

也可以像 border-radius 制作图片圆角的方法, 将图片转为容器 div 的背景图, 也能实现图 3-49 的效果, 但是会使用 JavaScript 脚本。对于不懂脚本的 Web 设计师来说, 还是蛮头痛的。具体的操作方法可以参考 border-radius 一节。

4. 多层阴影

前几种都是单阴影效果的使用, 其实 box-shadow 可以多层阴影同时使用, 每层阴影之间使用逗号 “,” 隔开。而每层阴影的使用方法都和前面一样, 例如:

```

.box-shadow {
    width: 200px;
    height: 100px;
    border: 1px solid #ccc;
    border-radius: 5px;
}

```

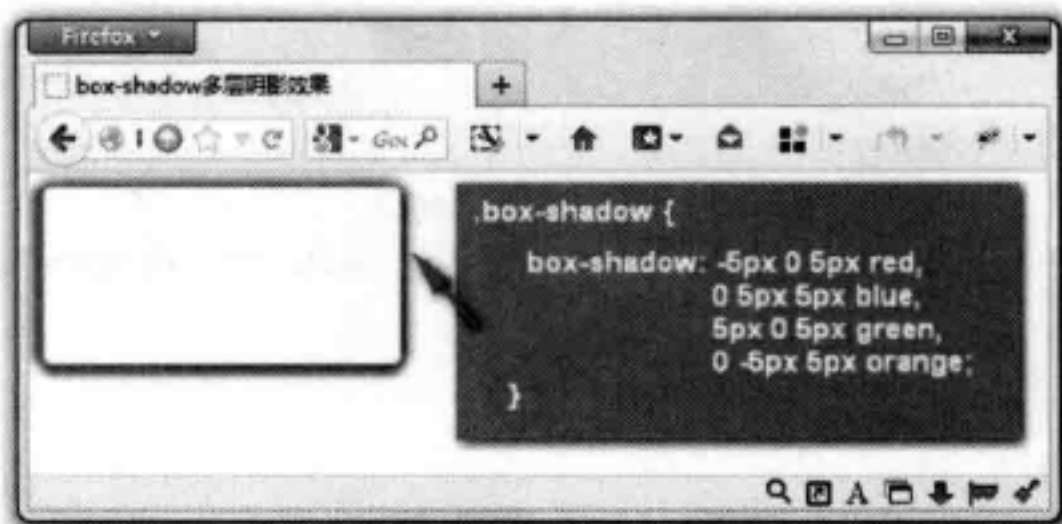


```
box-shadow: -5px 0 5px red, 0 5px 5px blue, 5px 0 5px green, 0 -5px 5px orange;
}
```

效果如图 3-50 所示。

制作多层阴影效果时，不设置模糊半径，只设置扩展半径，并配合多个阴影颜色，还可以制作多色边框效果，代替 border-color 属性制作多色边框效果，例如：

```
.box-shadow {
  width: 200px;
  height: 100px;
  border: 1px solid #ccc;
  margin: 30px;
  box-shadow: 0 0 0 1px red,
              0 0 0 5px blue,
              0 0 0 8px green,
              0 0 0 12px yellow,
              0 0 0 16px orange,
              0 0 0 20px #06c,
              0 0 0 24px lime;
}
```



☆图 3-50 box-shadow 多层阴影效果

效果如图 3-51 所示。

使用 box-shadow 制作多色边框效果，需要注意模仿 border 的宽度。前面介绍过 box-shadow 的工作模式，在计算宽度时需要减去前面阴影的值，才是显示的颜色宽度。

在使用多层级 box-shadow 时，还需要特别注意阴影的顺序，最先写的阴影将显示在最顶层，如上面的示例，先定义 1px 红色阴影，再定义 5px 蓝色阴影，接着是 8px 绿色阴影，以此类推。显示结果就是红色在蓝色上面，蓝色在绿色上面，绿色在黄色上面，以此类推。但是，如果最前面的阴影太大，顶层的阴影就会遮盖底部的阴影。例如，上例中将最底层的 24px 的 lime 阴影放到最前面，效果就完全不一样了。

```
.box-shadow {
  width: 200px;
  height: 100px;
  border: 1px solid #ccc;
  margin: 30px;
  box-shadow: 0 0 0 24px lime,
              0 0 0 1px red,
              0 0 0 5px blue,
              0 0 0 8px green,
              0 0 0 12px yellow,
              0 0 0 16px orange,
}
```

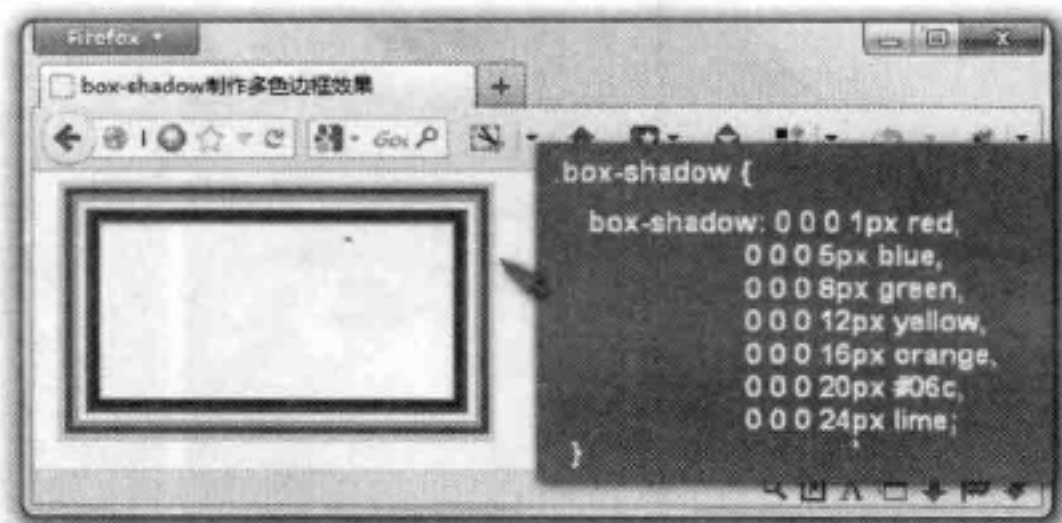


图 3-51 box-shadow 制作多色边框效果

```

0 0 0 20px #06c;
}

```

此时后面的阴影都被第一个阴影遮盖了,如图 3-52 所示。

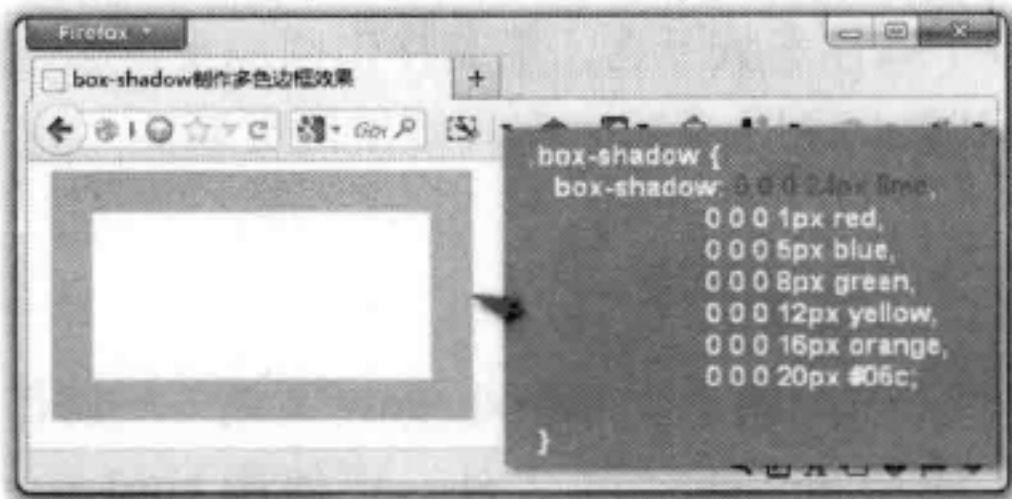







图 3-52 被遮盖的阴影效果

3.5.3 浏览器兼容性

目前 box-shadow 属性得到很好的支持,IE 8 及以前版本的浏览器不支持 box-shadow 属性。在现代浏览器的新版本中无须加各浏览器的前缀,不过要向前兼容,Firefox 3.5 ~ 3.6 下需要添加“-moz-”,Chrome 4 ~ 9 和 Safari 3.1 ~ 5.0 浏览器中添加“-webkit”。借助兼容方式,各主流浏览器对 box-shadow 属性的支持情况如表 3-7 所示。

表 3-7 box-shadow 的浏览器兼容表

属性名					
box-shadow	9 + √	3.5 + √	2.0 + √	10.5 + √	4.0 + √

虽然 IE 低版本不支持这个属性,但目前 box-shadow 在实际项目中运用越来越普遍。因为 box-shadow 实现阴影比使用背景图片的方法方便,同时能为 Web 前端设计师减少很多时间,维护也方便。

要兼容 IE 低版本,可以使用 IE 的滤镜来模拟实现。

```

filter: progid:DXImageTransform.Microsoft.Shadow(color='颜色值',
          Direction=阴影角度(数值), Strength=阴影半径(数值));

```

其中“DropShadow”(盒状阴影)和“Shadow”(阴影)两个滤镜正是为实现阴影而设,另外“Glow”(发光)滤镜则用于在盒容器四周实现发光阴影。但这些滤镜可设置的参数并不像 box-shadow 属性那样提供诸多的自定义参数,我不认为这些滤镜能够实现前面示例中所需的效果。当然除了 IE 滤镜之外,同样可以采用前面说的 PIE 和 IE -CSS3 脚本来实现 IE 下的阴影效果。



提示 现代浏览器使用 box-shadow 来制作阴影,而不支持 box-shadow 的浏览器让它不显示阴影,如果非要完美兼容,不妨考虑在不支持 box-shadow 的浏览器中使用背景图片来模仿阴影。

3.5.4 box-shadow 属性的优势

从实现盒子阴影来说，box-shadow 是最方便的，不管是使用背景图片，还是使用滤镜或者说 JavaScript 脚本，都无法与 box-shadow 属性相比。

- ❑ box-shadow 具有多个属性参数可选，能制作出圆润平滑的阴影效果。
- ❑ 代码维护方便，可以随时更改参数来实现效果的更新。

3.5.5 实战体验：制作 3D 搜索表单

为了方便读者理解，接下来介绍一个 box-shadow 案例——制作 3D 搜索表单。

在这个案例中，除了使用 box-shadow 之外，还使用 border-radius 制作圆角，并涉及 text-shadow 制作文本阴影，以及 gradient 制作渐变背景图片，关于这两项技术，请参阅后面章节。整个案例的效果如图 3-53 所示。

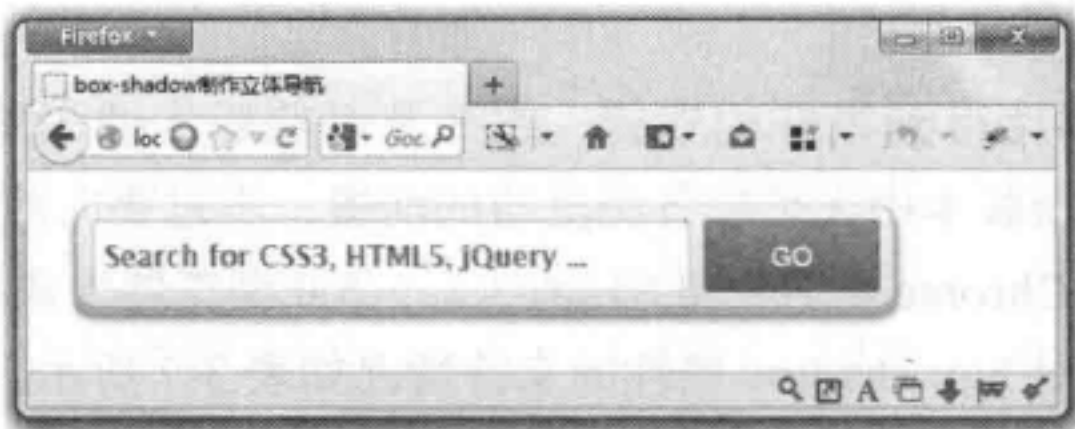


图 3-53 Box-shadow 制作 3D 搜索表单

从 box-shadow 多层次阴影特性出发，给表单容器设置多个同方向阴影效果，并且配合圆角属性 border-radius 来描绘圆角线框，同时使用渐变属性制作渐变的背景图片等，结合 CSS3 的多种效果，从而构建出这个 3D 搜索表单。

1. 构建 3D 表单的结构

整个表单结构很简单，代码如下所示。

```
<!-- 表单结构 -->
<form id="formWrapper">
  <div class="formFiled clearfix">
    <!-- 搜索表单的输入框 -->
    <input type="text" required="" placeholder="Search for CSS3, HTML5, jQuery ..."
class="search">
    <!-- 搜索按钮 -->
    <input type="submit" class="btn submit" value="go">
  </div>
</form>
```

整个结构使用一个“form”元素，并且应用一个“div.formFiled”容器来包裹“input.search”的输入框和一个搜索按钮“input.btn”，如图 3-54 所示。

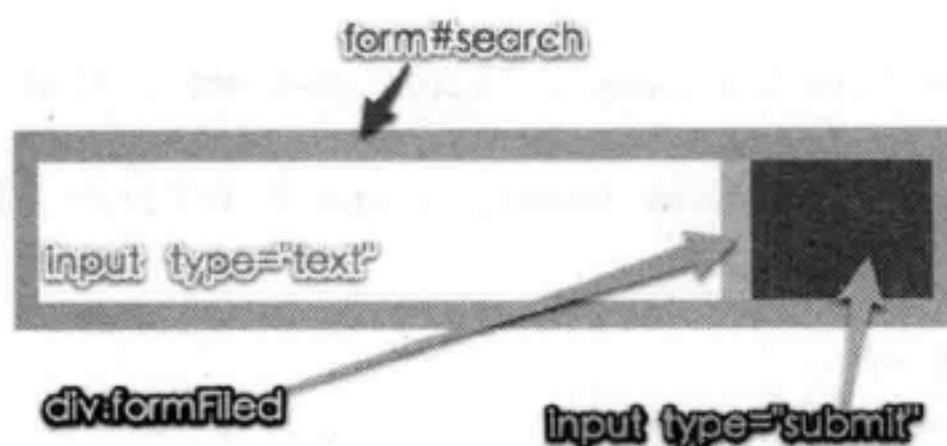


图 3-54 搜索表单结构

2. 设计表单容器的 3D 立体效果

使用 box-shadow 的多层阴影特性给表单元素设计 3D 立体效果，样式代码如下所示。

```
#formWrapper {
    width: 450px; /* 设置搜索表单的宽度 */
    padding: 8px;
    margin: 20px;
    overflow: hidden; /* 清除浮动 */
    /* 设置表单的边框效果 */
    border-width: 1px;
    border-style: solid;
    border-color: #dedede #bababa #aaa #bababa;
    /* 最为关键的代码，设置表单 3D 立体效果 */
    box-shadow: 0 3px 3px rgba(255,255,255,.1),
                0 3px 0 #bbb, 0 4px 0 #aaa,
                0 5px 5px #444;
    /* 设置圆角效果 */
    border-radius: 10px;
    /* 使用渐变制作表单的渐变背景图片 */
    background-color: #f6f6f6;
    background-image: -webkit-gradient(linear, left top,
                                      left bottom, from(#f6f6f6), to(#eae8e8));
    background-image: -webkit-linear-gradient(top, #f6f6f6, #eae8e8);
    background-image: -moz-linear-gradient(top, #f6f6f6, #eae8e8);
    background-image: -ms-linear-gradient(top, #f6f6f6, #eae8e8);
    background-image: -o-linear-gradient(top, #f6f6f6, #eae8e8);
    background-image: linear-gradient(top, #f6f6f6, #eae8e8);
}
```

3. 制作表单输入框的搜索按钮效果

接下来使用 box-shadow 制作 3D 立体效果，为了使表单更漂亮，将输入框和搜索按钮进行美化，代码如下所示。

```
/* 输入框样式效果 */
#formWrapper .search {
    width: 330px;
    height: 20px;
    padding: 10px 5px;
```

```

float: left;
font: bold 16px 'lucida sans', 'trebuchet MS', 'Tahoma';
border: 1px solid #ccc;
box-shadow: 0 1px 1px #ddd inset, 0 1px 0 #fff; /* 多阴影效果 */
border-radius: 3px;
}
/* 输入框得到焦点时样式 */
#formWrapper .search:focus {
outline: 0;
border-color: #aaa;
box-shadow: 0 1px 1px #bbb inset;
}
#formWrapper .search::-webkit-input-placeholder,
#formWrapper .search:-moz-placeholder,
#formWrapper .search:-ms-input-placeholder {
color: #999;
font-weight: normal;
}
/* 搜索按钮效果 */
#formWrapper .btn {
float: right;
border: 1px solid #00748f;
height: 42px;
width: 100px;
padding: 0;
cursor: pointer;
font: bold 15px Arial, Helvetica;
color: #fafafa;
text-transform: uppercase;
background-color: #0483a0;
background-image: -webkit-gradient(linear, left top,
                                left bottom, from(#31b2c3), to(#0483a0));
background-image: -webkit-linear-gradient(top, #31b2c3, #0483a0);
background-image: -moz-linear-gradient(top, #31b2c3, #0483a0);
background-image: -ms-linear-gradient(top, #31b2c3, #0483a0);
background-image: -o-linear-gradient(top, #31b2c3, #0483a0);
background-image: linear-gradient(top, #31b2c3, #0483a0);
border-radius: 3px;
text-shadow: 0 1px 0 rgba(0, 0, 0, .3);
box-shadow: 0 1px 0 rgba(255, 255, 255, 0.3) inset, 0 1px 0 #fff;
}
/* 按钮悬浮状态和焦点状态下效果 */
#formWrapper .btn:hover,
#formWrapper .btn:focus {
background-color: #31b2c3;
background-image: -webkit-gradient(linear, left top,
                                left bottom, from(#0483a0), to(#31b2c3));
background-image: -webkit-linear-gradient(top, #0483a0, #31b2c3);
background-image: -moz-linear-gradient(top, #0483a0, #31b2c3);
background-image: -ms-linear-gradient(top, #0483a0, #31b2c3);
background-image: -o-linear-gradient(top, #0483a0, #31b2c3);
background-image: linear-gradient(top, #0483a0, #31b2c3);
}

```

```
background-image: linear-gradient(top, #0483a0, #31b2c3);
}
/* 按钮点击时效果 */
#formWrapper .btn:active {
outline: 0;
box-shadow: 0 1px 4px rgba(0, 0, 0, 0.5) inset;
}
/*firefox 下按钮去除焦点线*/
#formWrapper::-moz-focus-inner {
border: 0;
}
```

制作的 3D 立体搜索表单效果如图 3-53 所示。当然这只是一个简单的案例，box-shadow 还可以制作更多的效果，例如双层边框效果、发光效果、立体按钮等。大家还可以发挥自己的想象力，创造出更多的有创意的、有吸引力的 UI 效果。

3.6 本章小结

本章主要介绍 CSS3 新增的边框特性，首先从 CSS 的 border 属性着手切入，分别介绍了 CSS3 新增边框特性，border-color、border-image、border-radius 以及 box-shadow。详细介绍了每个特性的语法规则，并且结合一些简单的案例，以图解的方式介绍了这些特性的具体使用方法以及在 IE 下相应的兼容和处理方式。

CSS3 背景

`background` 是 CSS 中使用频率很高的一个属性，可以帮助 Web 设计师实现一些特殊的效果，但有些时候 CSS 中提供的 `background` 功能远远无法满足设计的需求。基于这种情形之下，为了方便设计师更灵活地设计需要的网页效果，在原有 `background` 基础上新增了一些功能属性，可以在同一个元素内叠加多个背景，也允许设计师改变背景图片的大小尺寸，设计师还可以自己指定背景图显示的范围以及指定绘制背景图像的绘制起点等。本章主要围绕这几方面，向大家介绍 CSS3 中 `background` 的新特性。

4.1 CSS3 背景属性简介

`background` 是一个使用率很高的属性，也是一个十分有用的属性，能帮助设计师实现一些特殊的效果，使用起来也非常简单。

4.1.1 背景的基本属性

背景主要包括五个属性：

- ❑ `background-color`（背景颜色）
- ❑ `background-image`（背景图片）
- ❑ `background-repeat`（背景图片展示方式）
- ❑ `background-attachment`（背景图片是固定还是滚动）
- ❑ `background-position`（背景图片位置）

可以单独写，也可以将这些属性串在一起使用。

```
background: <background-color> [,<background-image>] [,<background-repeat>]
[,<background-attachment>] [,<background-position>]
```

1. background-color 属性

语法:

```
background-color: transparent || <color>
```

用来设置元素的背景颜色,其默认值为“transparent”,不设置任何颜色情况下是透明色,<color>用来设计背景色彩,常用的颜色格式如下。

- 颜色名:如“red”;
- rgb 色:如 rgb (255,0,0) 或 rgb (100%,0%,0%);
- hls 值:如 hsl (0,100%,50%),
- 十六进制值:如 #ff0000。

在 CSS3 中还可以使用 rgba 色,如 rgba (255,0,0,0.3) 可以使用 hsla 值,如 hsla (0,100%,50%,0.5),有关于这两个颜色值的使用,请参阅后面的章节。

2. background-image 属性

语法:

```
background-image: none || <url>
```

用来设置元素的背景图片,默认值为“none”,<url>是指背景图片的地址,这个地址可以是相对地址,也可以是绝对地址。

3. background-repeat 属性

语法:

```
background-repeat: repeat || repeat-x || repeat-y || no-repeat
```

用来设置元素背景图片在元素的盒模型中的铺放格式,其默认为“repeat”,也就是背景图片沿元素的 X 轴和 Y 轴同时平铺,“repeat-x”表示的是元素背景图片沿元素的 X 轴平铺,Y 轴不进行任何铺放;“repeat-Y”刚好相反,元素背景图片沿元素的 Y 轴平铺,X 轴不进行任何铺放;“no-repeat”和默认值“repeat”相反,表示背景图片不做任何铺放。

4. background-attachment 属性

语法:

```
background-attachment: scroll || fixed
```

用来设置元素背景图片是否固定或者随着页面的其余部分滚动,其默认值为“scroll”,表示背景图片会随着浏览器滚动条一起滚动,而取值为“fixed”时,背景图片固定不动。



注意 background-attachment 取值为“fixed”时,一般运用在 html 或 body 标签上,使用在其他标签上不能达到固定效果。

5. background-position 属性

语法：

background-position: <percentage>||<length>||[left|center|right] [,top|center|bottom]

用来设置元素背景图片的位置，其默认值为 (0, 0) || (0%, 0%) || (left top)，其值可以是具体的百分数或数值设置 (可以是负值)，也可以使用关键词 left、center、top、right、top、bottom 配合设置，而以下几种方式表示的是背景图片相同的定位方式，如图 4-1 所示。

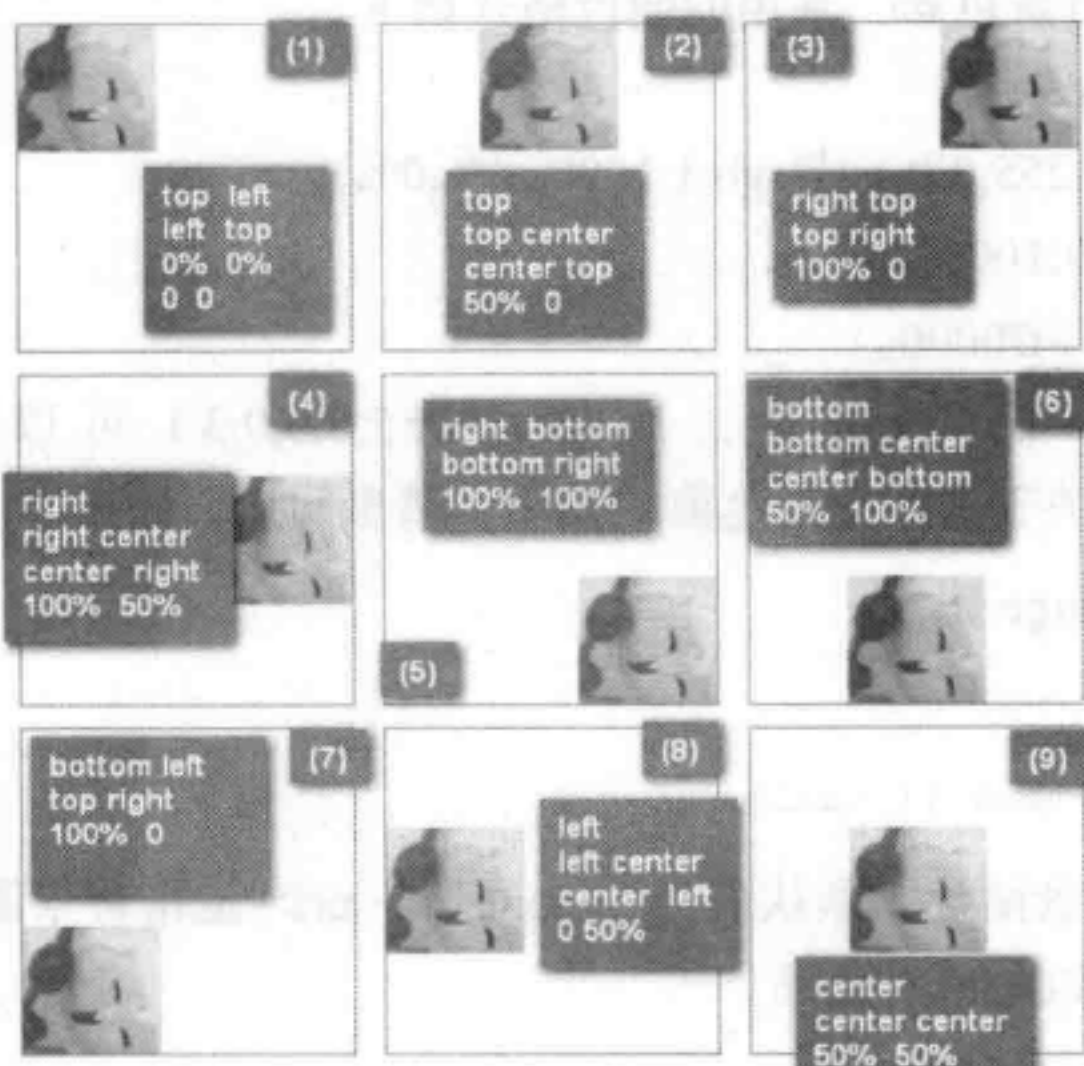


图 4-1 background-position 演示效果

(1) “top left”、“left top” 和 “0% 0%”、“0 0”

表示元素背景图片的起点位置与元素的左上角吻合。

(2) “top”、“top center”、“center top” 和 “50% 0”

表示元素背景图片的顶边中心点与元素的顶边中心点吻合。

(3) “right top”、“top right” 和 “100% 0”

表示元素背景图片的右上顶点与元素右上顶点吻合。

(4) “right”、“right center”、“center right” 和 “100% 50%”

表示元素背景右边中心点与元素右边中心点吻合。

(5) “bottom right”、“right bottom” 和 “100% 100%”

表示元素背景图右下顶角与元素右下角吻合。

(6) “bottom”、“bottom center”、“center bottom” 和 “50% 100%”

表示元素背景图片底边中心点与元素底边中心点吻合。

(7) “bottom left”、“left bottom” 和 “0% 100%”

表示元素背景图片左下顶角与元素左下顶角吻合。

(8) “left”、“left center”、“center left” 和 “0% 50%”

表示元素背景图片左边中心点与元素左边中心点吻合。

(9) “center”、“center center” 和 “50% 50%”

表示元素背景图片正中心点与元素正中心点吻合。

4.1.2 与背景相关的新增属性

关于 background 属性的用法相信很多读者都已经熟悉了。在 CSS3 中, background 属性依然保持以前的用法, 只是追加了一些与背景相关的属性。

- background-origin: 指定绘制背景图片的起点。
- background-clip: 指定背景图片的显示范围。
- background-size: 指定背景图片的尺寸大小。
- background-break: 指定内联元素的背景图片进行平铺时的循环方式。

4.2 CSS3 背景原点属性

background-origin 是 background 新增属性之一, 本节以及后面三节的将详细介绍其他属性的用法。

4.2.1 background-origin 属性的语法及参数

background-origin 属性主要用来决定 background-position 属性的参考原点, 即决定背景图片定位的起点。在默认情况下, 背景图片的 background-position 属性总是以元素左上角为坐标原点对背景图片进行背景定位。CSS3 的 background-origin 属性将打破这一格局, 可以根据自己的需要来改变背景图片的 background-position 起始位置。先来学习其基本语法。

```
background-origin: padding || border || content
```

这种语法是早期的 Webkit 和 Gecko 内核浏览器 (Firefox 3.6-、Safari 和 Chrome 低版本) 支持的一种老的语法, 在新版本浏览器下, background-origin 具有一种新的语法如下。

```
background-origin: padding-box || border-box || content-box
```



注意 IE 9+、Firefox 4+、Chrome 4+、Safari 3+ 和 Opera10.5+ 都支持新的语法格式。

background-origin 具有三个属性值: padding-box、border-box 和 content-box。接下来简单说明这三个属性值的作用。

- padding-box(padding): 默认值, 决定 background-position 起始位置从 padding 的外

边缘 (border 的内边缘) 开始显示背景图片。

❑ border-box(border) : 决定 background-position 起始位置从 border 的外边缘开始显示背景图片。

❑ content-box(content) : 决定 background-position 起始位置从 content 的外边缘 (padding 的内边缘) 开始显示背景图片。



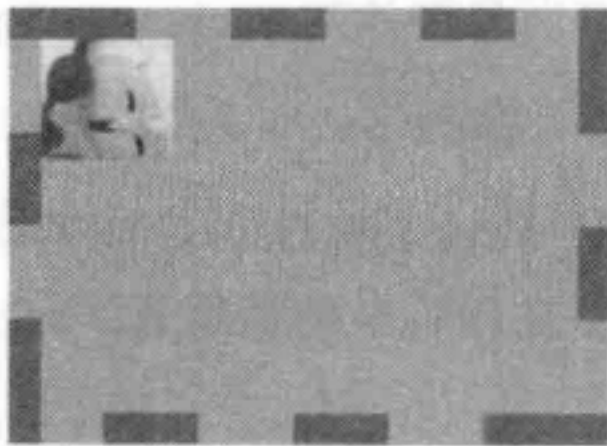
注意 IE 8 下以版本 background-origin 的默认值 border, 背景图片的 background-position 是从 border 开始显示背景图片。

4.2.2 background-origin 属性使用方法

本节通过几段简单的代码来辅助我们理解 background-origin 的三个属性值的实际使用, 以及它们之间有何区别。

首先按照常规的方法创建一个 div, 并且设置这个 div 的相关样式, 例如案例中设置了元素 div 的边框为 20px 的点线状, 20px 的内距 padding 以及背景色为橙色, 同时给其一张不重复铺张的背景图, 详细代码如下所示。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>background-origin 使用方法 </title>
  <style type="text/css">
    div {
      width: 300px;
      height: 200px;
      border: 20px dashed rgba(0,0,0,0.3);
      background: orange url(bg.png) no-repeat left top;
      padding: 20px;
      margin: 30px;
    }
  </style>
</head>
<body>
  <div class="padding-box"></div>
</body>
</html>
```



```
div {
  width: 300px;
  height: 200px;
  border: 20px dashed
  rgba(0,0,0,0.3);
  background: orange
  url(bg.png) no-repeat left top;
  padding: 20px;
  margin: 30px;
}
```

初步效果如图 4-2 所示。

图 4-2 div 默认背景图片设置效果

上面例子是给元素设置背景图片的常用方法, 接下来依次修改元素的 background-origin 值。

1. padding-box

首先在刚才的 div 元素上, 显式设置 background-origin 的值为 padding-box, 如下所示。

```
.padding-box {
    -webkit-background-origin: padding-box;
    -moz-background-origin: padding-box;
    -o-background-origin: padding-box;
    -ms-background-origin: padding-box;
    background-origin: padding-box;
}
```

如果把 background-origin 设置为 padding-box 时, 效果和图 4-2 所示效果完全一样。因为元素的 background-origin 默认属性值就是 padding-box, 换句话说, 前面虽然没有显式设置 background-origin 属性值, 但是浏览器会将元素的 background-origin 设置为 padding-box。也就是说背景图片 background-position 起始点是从元素的边框内边缘 (padding 的外边缘) 开始, 如图 4-3 所示。

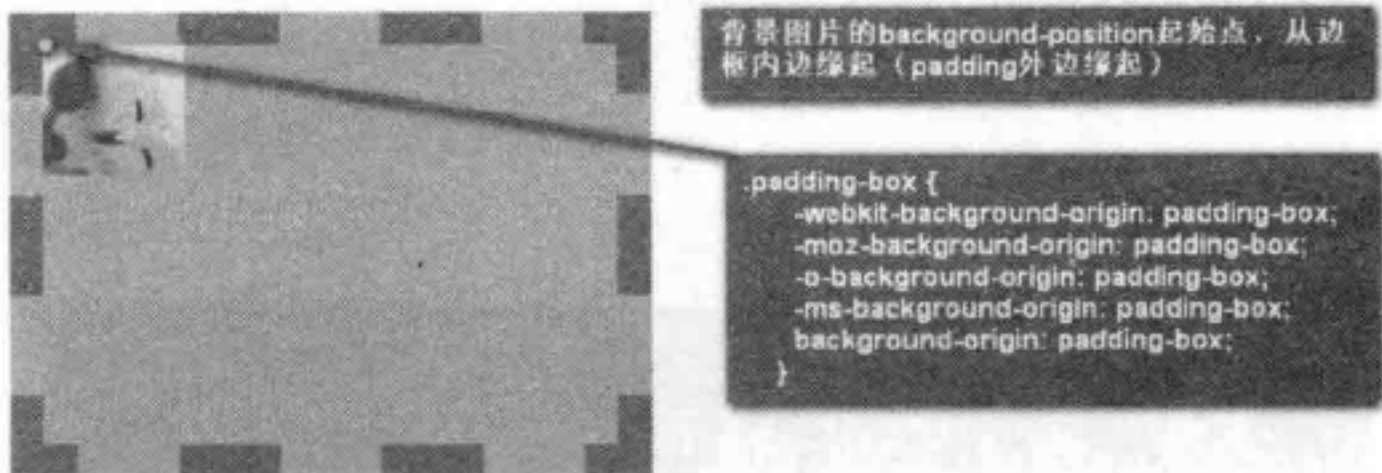


图 4-3 元素设置 background-origin 为 padding-box 的效果

2. border-box

在前面的实例基础上, 将元素 div 的 background-origin 属性值变为 border-box, 如下所示。

```
.border-box {
    -webkit-background-origin: border-box;
    -moz-background-origin: border-box;
    -o-background-origin: border-box;
    -ms-background-origin: border-box;
    background-origin: border-box;
}
```

将 background-origin 值变为 border-box 后的效果如图 4-4 所示。

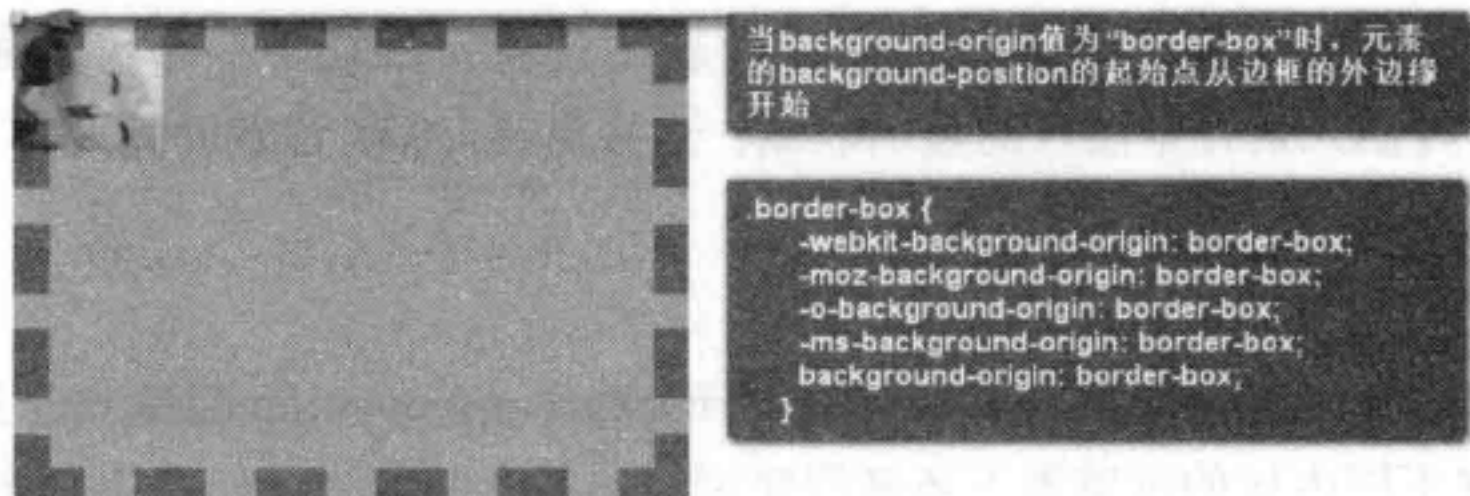


图 4-4 元素设置 background-origin 值为 border-box 的效果

图 4-4 与图 4-3 相比较, 把元素 div 的 background-origin 值设置为 border-box 时, 不难发现, 元素 div 的背景图片的 background-position 起始位置不在从元素边框内边缘 (padding 外边缘) 开始, 而是变成了从元素边框外边缘开始, 此时的背景图片直接在元素的边框底下。

3. content-box

background-origin 值为 padding-box 时改变了元素 background-position 的起始点位置, 将 background-origin 值换成 content-box 又将会有什么样的变化呢? 先通过实例来演示一下。

```
.content-box {
    -webkit-background-origin: content-box;
    -moz-background-origin: content-box;
    -o-background-origin: content-box;
    -ms-background-origin: content-box;
    background-origin: content-box;
}
```

此时效果如图 4-5 所示。

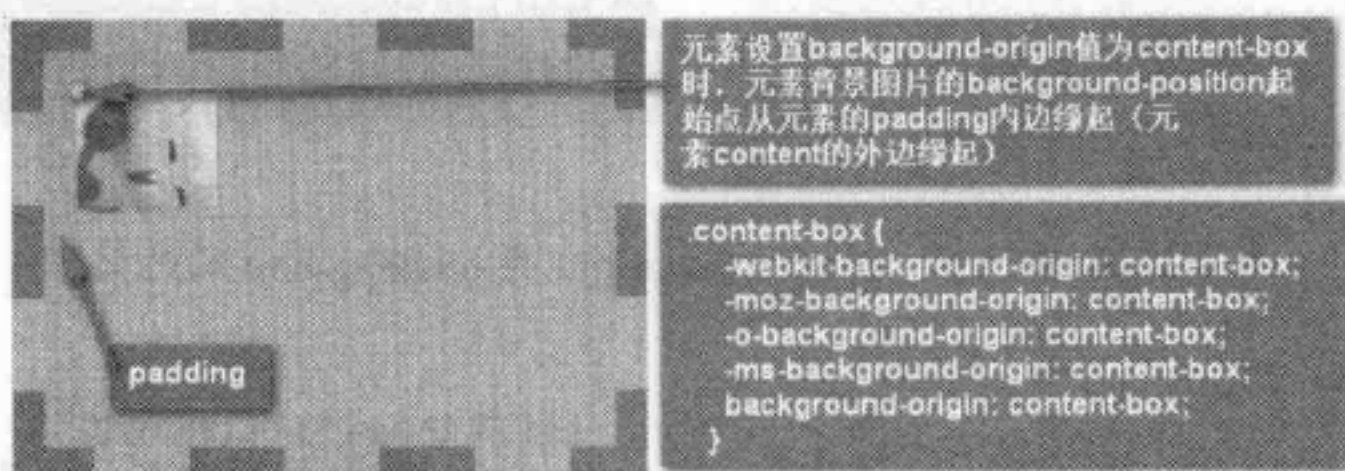


图 4-5 元素设置 background-origin 值为 content-box 的效果

图 4-5 所示的效果正如前面介绍的一样, 此时元素的 background-position 起始点位置从元素的 padding 内边缘 (元素 content 外边缘) 开始。

background-origin 属性改善了背景图片定位的方式, 使 Web 前端设计师能够更灵活地决定背景图片应该显示的位置。但大家发现没有, 上面三个实例都是 background-attachment 为 scroll, 如果将 background-attachment 设置为 fixed, background-origin 将不起任何作用。

background-origin 改变了元素背景图片的起始位置, 配合下一节介绍的 background-clip 属性, 可以控制元素背景图片的显示区域, 这将更显 CSS3 背景的魅力。

4.2.3 浏览器兼容性

background-origin 属性能决定背景图片的 background-position 起始点, 虽然现代浏览器都支持, 但在不同内核的浏览器下还是需要浏览器各自的前缀, 如表 4-1 所示。

表 4-1 background-origin 各浏览器的私有属性

	Mozilla Gecko	Webkit	Presto	Konqueror	Internet Explorer
background-origin	-moz-	-webkit-	-o-	-khtml-	-ms-

为了保证只要支持 background-origin 属性的浏览器都能正常运行，就需要按下面的方式使用。






```

/*Old Webkit and Gecko*/
-moz-background-origin: padding || border || content;
-webkit-background-origin: padding || border || content;
/*New Webkit and Gecko*/
-moz-background-origin: padding-box || border-box || content-box;
-webkit-background-origin: padding-box || border-box || content-box;
/*Presto*/
-o-background-origin: padding-box || border-box || content-box;
/*W3c 标准*/
background-origin: padding-box || border-box || content-box;

```

借助兼容方式，各主流浏览器对 background-origin 属性的支持情况如表 4-2 所示。

表 4-2 background-origin 浏览器兼容性

属性名					
background-origin	9 + √	3 + √	1.0 + √	10.5 + √	3.1 + √

4.3 CSS3 背景裁切属性

background-clip 属性是 background 新增的第二个属性，用来定义背景图像的裁剪区域。和 background-origin 属性有几分相似，目前这个属性可以接受的值有：

- ☐ padding-box（背景延伸到 padding 的外边缘，但不会超出边框的范围）；
- ☐ border-box（背景图片在边框下，这个也是 background-clip 的默认值）；
- ☐ content-box（背景仅在内容区域绘制，不会超出 padding 和边框的范围）。

4.3.1 background-clip 属性的语法及参数

在学习这个属性之前，首先看它的语法规则。

```
background-clip : border-box || padding-box || content-box
```

background-clip 的语法规则和 background-origin 语法规则一样，其取值也是相似，但 background-clip 在 Gecko 内核浏览器（Firefox3.6 及之前的版本）不支持 content-box，并且使用 border 和 padding 来代替标准语法中的 border-box 和 padding-box，其语法如下。

```
background-clip : border || padding
```


Safari 5 能够在标准属性中支持 border-box 和 padding-box 属性值，但 content-box 属性值也需要加上 Webkit 内核前缀“-webkit-”。在 Webkit 内核浏览器下还支持一个特殊值 text，但是在 -webkit- 前缀的属性里，这个值可以让文本作为背景的遮罩，不在文本之后的背景则不可见。这个属性使用率极低。

background-clip 和 background-origin 属性值相似，属性参数的含义如下。

- ❑ border-box：默认值，元素背景图像从元素的 border 区域向外裁剪，即元素边框之外的背景图片都将被裁剪掉。
- ❑ padding-box：元素背景图像从 padding 区域向外裁剪，即元素 padding 区域之外的背景图像将被裁剪掉。
- ❑ content-box：元素背景图像从 content 区域向外裁剪，即元素内容区域之外的背景图像将被裁剪掉。

在 HTML 页面中，一个具有背景的元素通常由元素的内容（content）、内部补白（padding）、边框（border）和外部补白（margin）四部分构成。这也是 CSS 中所说的盒子模型，如图 4-6 所示。

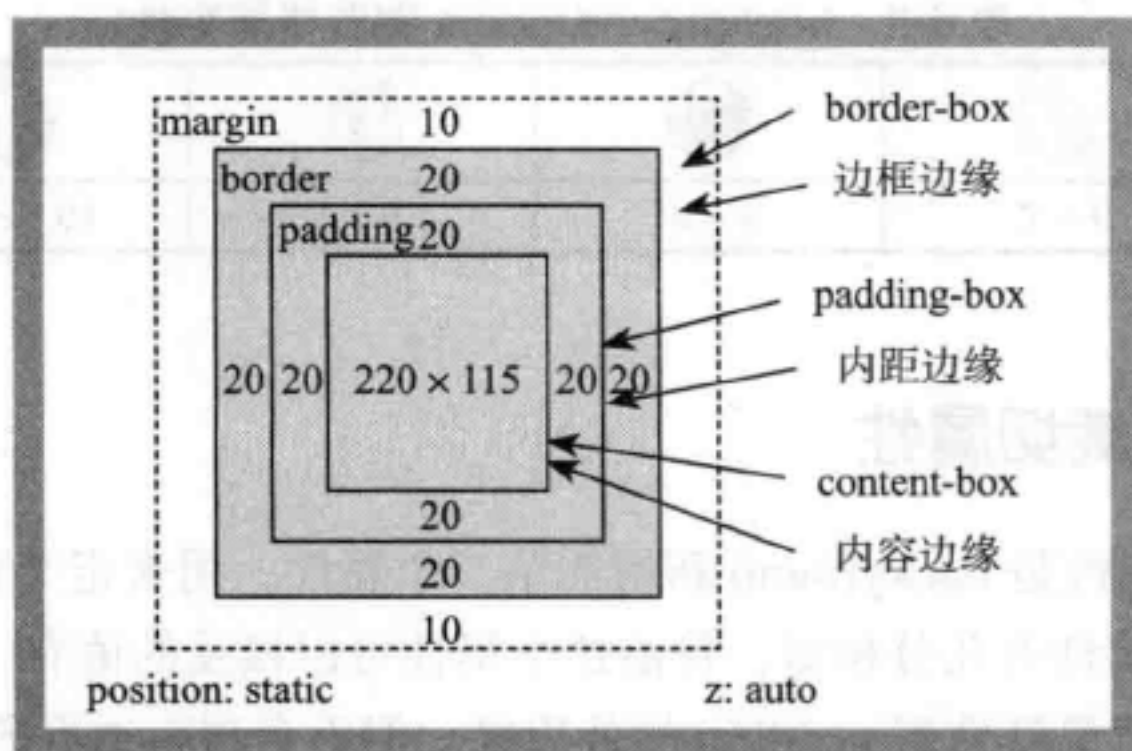


图 4-6 具有背景的元素盒子模型

CSS2 元素背景的显示范围与 CSS2.1、CSS3 并不相同。在 CSS2 中，背景的显示范围是指内部补白（padding）之内的范围，不包括边框；而在 CSS2.1 至 CSS3 中，背景在整个盒模型中，它是布满整个元素的盒子区域的，并不是从内部补白（padding）开始，只不过边框部分遮住了部分 background，但有一点需要注意，当元素的 background-repeat 为 no-repeat 时，background-color 是从元素的边框左上角起到边框右下角止，而 background-image 却不一样，从内部补白（padding）边缘的左上角起到元素边框（border）的右下角边缘止；当元素 background-repeat 的值为 repeat 时，background-color 此时完全在 background-image 之下，而且 background-image 从元素边框左上角起到元素边框右下角止，但 background-position 的起始点却是从元素的内部补白（padding）外边缘开始。如果对这段话不太好理解的话，先来看一个简单的实例。


```

<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>background 的默认显示</title>
  <style type="text/css">
    div {
      width: 300px;
      height: 100px;
      border: 50px dashed rgba(0,0,0,0.8);
      background: rgba(0,25,25,0.8) url(border.jpg) no-repeat;
      padding: 30px;
      margin: 20px;
    }
    .no-repeat {
      background-repeat: no-repeat;
    }
    .repeat {
      background-repeat: repeat;
    }
  </style>
</head>
<body>
  <div class="no-repeat"></div>
  <div class="repeat"></div>
</body>
</html>

```

此时效果如图 4-7 所示。

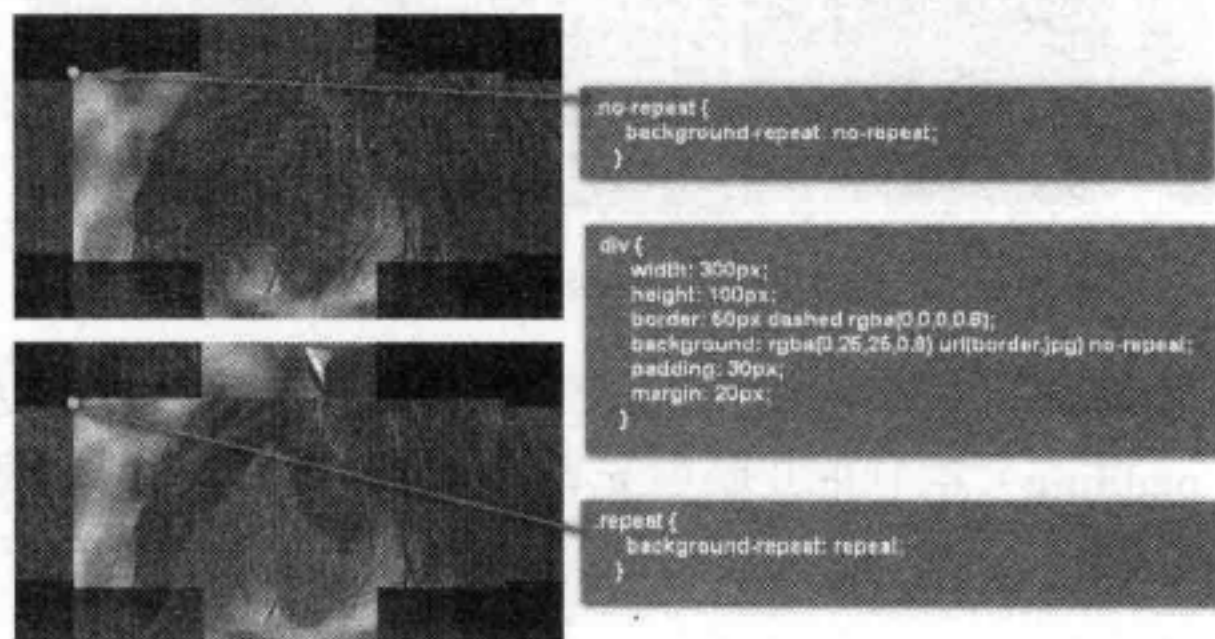


图 4-7 元素 background 默认显示风格

4.3.2 background-clip 属性使用方法

为了更直观地说明问题，通过实例来说明 background-clip 的使用方法。这个示例中具有三个 div 元素，设置它们的背景颜色均为橙色，边框为透明的紫色点划线，并且设置具有一个 30px 内部补白（padding）。在样式代码中指定这三个 div 元素的 background-clip 的属

性值分别为 border-box、padding-box 和 content-box，分别来看看这三个 div 元素在显示上有什么区别。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>background-clip 使用方法 </title>
  <style type="text/css">
    div {
      width: 200px;
      height: 100px;
      border: 30px dashed rgba(25,25,125,0.5);
      background: orange url(border.jpg) no-repeat;
      margin: 20px;
      padding: 30px;
    }
  </style>
</head>
<body>
  <div class="border-box"></div>
  <div class="padding-box"></div>
  <div class="content-box"></div>
</body>
</html>
```

在还没有给元素使用 background-clip 时默认效果如图 4-8 所示。



图 4-8 未显式设置 background-clip 属性的效果

图 4-8 再次证明了元素的背景填充整个元素，其中背景颜色填充到边框底下，元素的背景图片从内部补白（padding）左上角开始到元素边框右下角止。接下来，分别给元素添加对应的 background-clip 属性，从实际效果来区分它们之间的不同。

1. background-clip: border-box

首先将 background-clip 的值显式设置为 border-box，如下所示。

```
.border-box {
  -webkit-background-clip: border-box;
  -moz-background-clip: border-box;
  -o-background-clip: border-box;
  -ms-background-clip: border-box;
  background-clip: border-box;
}
```

`background-clip` 的默认值是 `border-box`，虽然在元素上显式设置 `background-clip` 的属性值为 `border-box`，但其效果和没有设置将是一样的，元素 `div` 背景色在边框底下，元素 `div` 的背景图片从元素左上角内部补白（`padding`）开始至右下角边框外边缘止，当然背景图片为 `repeat` 时，元素 `div` 的背景图片会铺满整个元素，如图 4-9 所示。

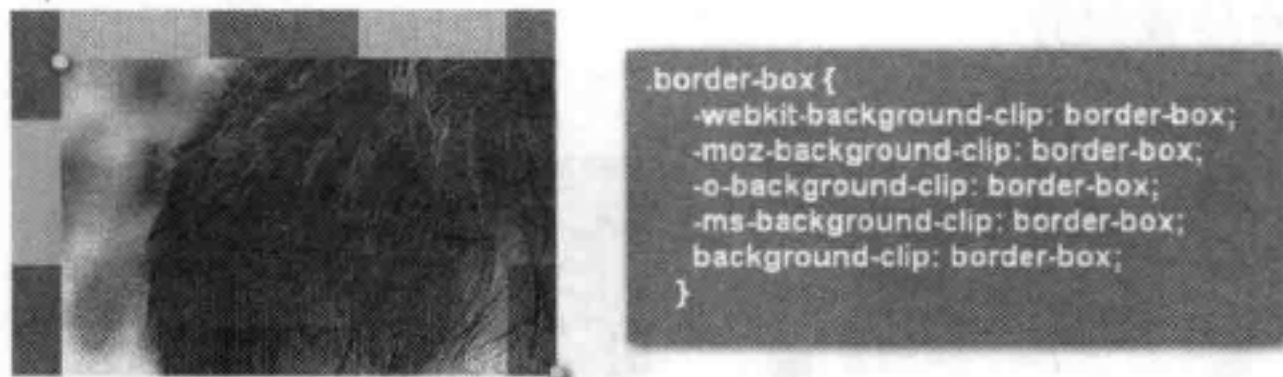


图 4-9 `background-clip` 值为 `border-box` 的效果

2. `background-clip:padding-box`

按同样的方法，在元素上设置 `background-clip` 的属性值为 `padding-box`。

```
.padding-box {
  -webkit-background-clip: padding-box;
  -moz-background-clip: padding-box;
  -o-background-clip: padding-box;
  -ms-background-clip: padding-box;
  background-clip: padding-box;
}
```

先来看效果的变化，如图 4-10 所示。

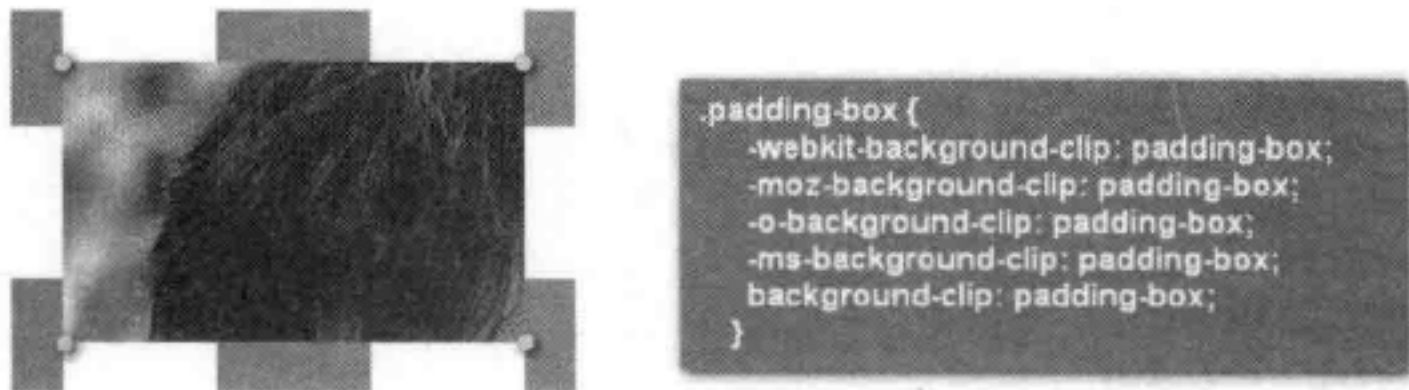


图 4-10 `background-clip` 的属性设置为 `padding-box` 的效果

`background-clip` 设置为 `padding-box` 值时，元素 `div` 的背景发生很大的变化，整个背景（背景色和背景图片）在超过 `padding` 外边缘的部分全部被裁剪掉了，此时并不是成比例裁剪，而是直接将超出 `padding` 边缘的背景剪掉。

3. `background-clip:content-box`

按同样的方法，把 `background-clip` 的属性值设置为 `content-box`，如下所示。

```
.content-box {
  -webkit-background-clip: content-box;
  -moz-background-clip: content-box;
}
```



```

-o-background-clip: content-box;
-ms-background-clip: content-box;
background-clip: content-box;
}

```

元素这个时候的背景图片效果如图 4-11 所示。元素 div 的背景只在内容区域显示，超过内容边缘的背景直接被裁剪。

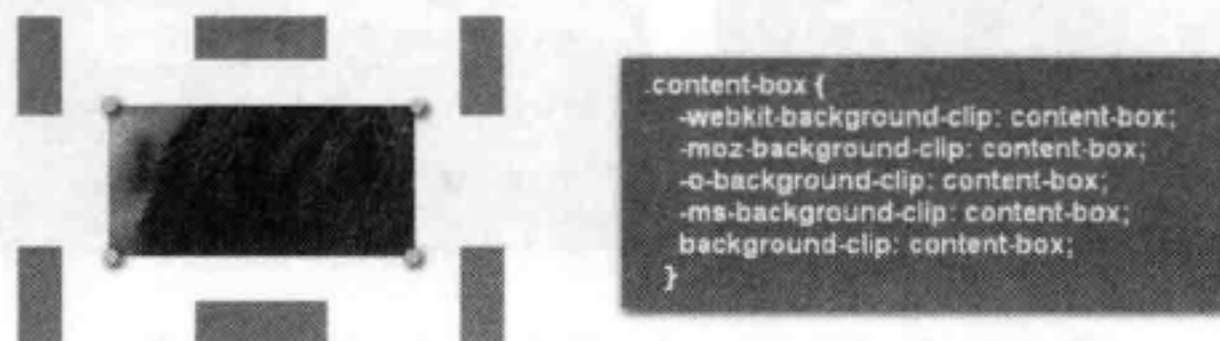


图 4-11 background-clip 设置为 content-box 的效果

4. text

通过三个简单的案例介绍了 background-clip 各个属性值的使用方法以及产生的效果，而 Webkit 内核下，background-clip 还有一个 text 属性，配合 Webkit 内核的私有属性 text-fill-color:transparent 可以制作背景图片填充文本的效果，一起看一个例子。

```

<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title> 背景图片填充文本效果 </title>
  <style type="text/css">
    div {
      width: 350px;
      background: orange url(border.jpg);
      padding: 20px;
      color:#fff;
      font-size: 60px;
      font-weight: bold;
      -webkit-background-clip: text;
      -webkit-text-fill-color: transparent;
    }
  </style>
</head>
<body>
  <div> 背景图片填充文本效果 <br />BACKGROUND</div>
</body>
</html>

```

背景图片填充了整个文本，效果如图 4-12 所示。

使用这种方法制作文本特效很炫，可惜目前只有 Webkit 浏览器支持。



图 4-12 背景图片填充文本的效果

4.3.3 浏览器兼容性






background-clip 在浏览器的兼容性方面和 background-origin 也极其相似，在使用时也需要添加各浏览器的前缀名，如表 4-3 所示。

表 4-3 background-clip 各浏览器的私有属性前缀

	Mozilla Gecko	Webkit	Presto	Konqueror	Internet Explorer
background-clip	-moz-	-webkit-	-o-	-khtml-	-ms-

借助兼容方式，各浏览器对 background-clip 属性的支持情况如表 4-4 所示。

表 4-4 各浏览器对 background-clip 的支持情况

属性名					
background-clip	9 + √	4 + √	4.0 + √	10.5 + √	5 + √

扩展阅读 background-origin 与 background-clip 属性的区别

从上两节的内容可以得知，background-origin 与 background-clip 属性实现的效果类似。但它们实现的原理却是不同的。在具体设计中，设计师选择使用 background-origin 属性还是 background-clip 属性还是需要考虑清楚，为了能更好地灵活使用这两个属性，大家需要对这两者相同与不同之处有所了解。

在了解它们区别之处之前，先看图 4-6 展示的元素盒模型的基本结构。对于任何一个元素来说，都具有四个区域（外补白区 margin、内补白区 padding、边框区 border 和内容区 content）和四个边缘线（外补白边缘、内补白边缘、边框边缘和内容边缘）。理解和掌握了这四区、四边缘就能更好地使用。

对于 background-origin 和 background-clip 属性而言，它们具有相同的三个属性值：border-box、padding-box 和 content-box。两个属性都可以设置多个属性值，多个属性值之间需要使用逗号隔开。但两者的虽具有相同的属性值，但其使用方法却不一样。

□ background-origin 属性是用来控制元素背景图片定位点（background-position）的起始位置。

□ background-clip 属性是用来控制元素背景图片（background-image）的展示区域。

对于 background-origin 属性来说，如果取值为 padding-box，背景图片起始位置

(background-position) 将相对于内补白 (padding) 的外边缘进行定位。其中, 当 background-position 属性值为 “0 0” 时, 背景图片定位起始点在内补白边缘的左上角处; 如果 background-position 取值为 “100% 100%” 时, 背景图片定位起始点在内补白边缘的右下角处。如果取值为 border-box 时, 此时元素背景图片起始定位点 (background-position) 将相对于元素边框 (border) 的外边缘进行定位, 同样定位点能确定图片起始点在边框左上角或者右下角处。取值为 content-box, 元素背景图片的定位起始点 (background-position) 则相对于内容外边缘 (content) 进行定位。

对于 background-clip 属性而言, 如果取值为 padding-box, 元素的背景图片 (background-image) 将忽略内补白 (padding) 外边缘, 此时元素的边框要是没有设置颜色, 将显示为透明, 换句话说, 整个背景图片只会在内补白和内容区域显示。取值为 border-box 时, 元素的背景图片 (background-image) 将忽略元素的边框 (border) 外边缘, 此时整个背景图片将显示在元素的边框底部 (也就是大家常见的元素背景展示风格, 整个背景图片在除外补白 margin 区域外全部展示)。如果取值为 content-box, 整个背景图片只显示在元素的内容 (content) 区域 (元素内补白和边框区域将不显示背景图片)。

建议通过上面的实例演示一回, 增强对这两个属性的理解, 做出具体的分析。

4.4 CSS3 背景尺寸属性

在 CSS3 中, 可以使用 background-size 属性来指定背景图片的尺寸, 可以控制背景图片在水平和垂直两个方向的缩放, 也可以控制图片拉伸覆盖背景区域的方式, 甚至还可以截取背景图片。背景图片能够自适应元素盒子的大小, 实现与模块大小完全适应的背景图片, 避免了因区块尺寸不同而需要设计不同的背景图片。

4.4.1 background-size 属性的语法及参数

background-size 属性的出现, 节省了前端人员很多时间。同样, 先从其语法入手。

```
background-size: auto || <length> || <percentage> || cover || contain
```

语法和 background 其他属性几乎一样, 只是其属性值不一样, 而且不同的属性值将会有截然不同的效果。接下来, 一起来看看 background-size 对应的属性参数起什么作用。

background-size 共有五种属性值, 每一种属性值的作用如下。

- ❑ auto: 默认值。将保持背景片的原始高度和宽度。
- ❑ <length>: 取具体的整数值 (例如 px 值), 将改变背景图片的大小。
- ❑ <percentage>: 取值为百分值, 可以是 0% ~ 100%。此时, 同样改变背景图片的大小, 但此值是相对于元素的宽度来进行计算, 并不是根据背景图片的宽度来进行计算。
- ❑ cover: 将背景图片放大, 以适合铺满整个容器。但这种方法会致使背景图片失真。

□ **contain** : 保持背景图像本身的宽高比例, 将背景图像缩放到宽度或高度正好适应所定义背景容器的区域。

当 **background-size** 取值为固定数值 (**length**) 和百分比值 (**percentage**) 时可以设置两个值, 也可以设置一个值。只取一个值时, 指定了背景图片的宽度, 第二个值相当于 **auto**, 也就是指定了高度。在这种情况下, **auto** 值设定之后能够让背景图片的高度自动地按照比例缩放。

4.4.2 background-size 属性使用方法

为了更好地了解 **background-size** 的工作机制, 接下来看几个简单的例子。

假设有一张宽 100px、高 50px 的背景图片 (如图 4-13 所示), 同时有一个宽 320px、高 270px 的元素 **div**, 将背景图片放置在这个元素中。

先创建一个文件, 其代码如下所示。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>background-size 的运用 </title>
  <style type="text/css">
    div {
      width: 320px;
      height: 270px;
      border: 1px solid red;
    }
  </style>
</head>
<body>
  <div></div>
</body>
</html>
```



图 4-13 示例背景图

在没有使用任何背景图片时的效果如图 4-14 所示。

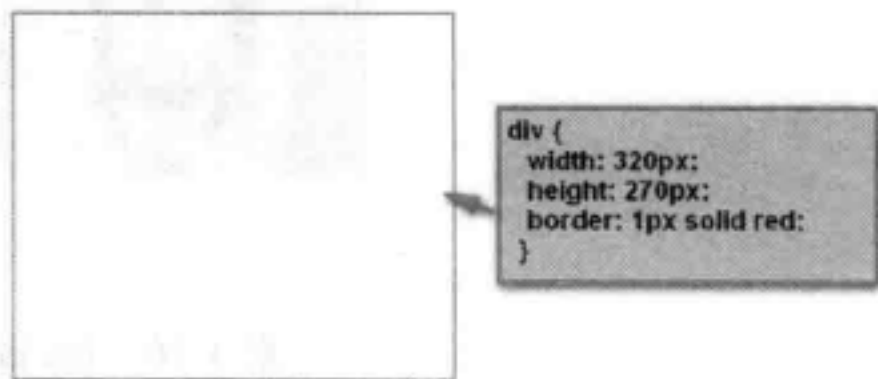


图 4-14 无背景图元素效果

1. auto

首先来看第一种效果, 当 **background-size** 取值为 **auto** 时, 代码如下。

```
div {
  width: 320px;
  height: 270px;
  border: 1px solid red;
  background: url(bg.jpg) no-repeat;
  background-size: auto;
}
```

此时元素 DIV 加上一个背景 bg.jpg，效果如图 4-15 所示。

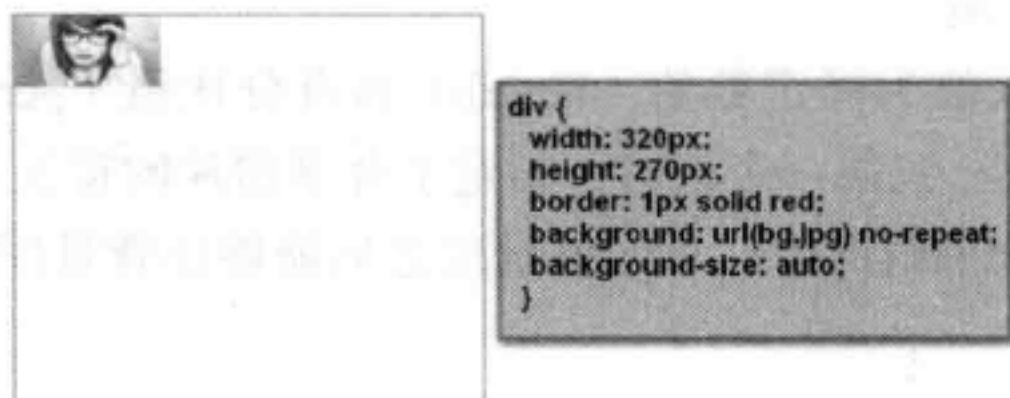


图 4-15 background-size 取值为 auto 的效果

图 4-15 说明，当 background-size 取值为 auto 时，背景图片没有做出任何的变化。前面也说过，auto 值就是使用背景图片保持原样，这个效果相当于没加 background-size，因为 background-size 在不显式设置时，元素默认就是 auto。

看第二种情况，在前面的基础上做一定的调整，把 background-size 的 auto 值换成固定的像素值，例如第一个值为 280px，第二个值为 200px，代码如下所示。

```
div {
    width: 320px;
    height: 270px;
    border: 1px solid red;
    background: url(bg.jpg) no-repeat;
    background-size: 280px 200px;
}
```

此时的效果如图 4-16 所示。

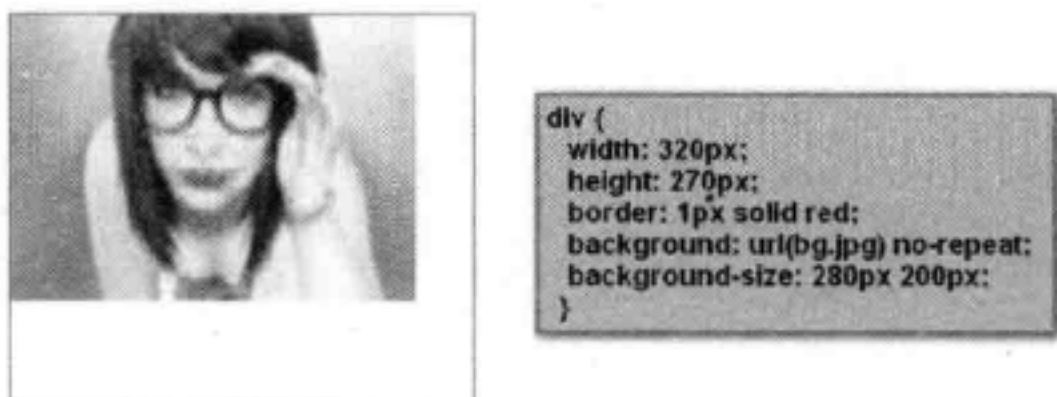


图 4-16 background-size 取固定像素值的效果

这个时候元素 div 的背景图片 bg.jpg 不是默认尺寸了，而是宽为 280px、高为 200px，同时背景图片由于拉伸造成了失真。

background-size 只取一个值时会是什么样呢？在前面的实例上，把 background-size 的第二个属性值 200px 去掉，此时 background-size 就相当于“280px auto”，也就是说背景图片的宽度依然是 280px，但背景图片的高度就不再是 200 像素，而是根据背景图片的宽度做了一定的比例计算。实际效果如图 4-17 所示。

2. percentage

除了浮点数字和单位标识符组成的长度值之外，background-size 还可以使用 0% ~ 100%

的值。当 `background-size` 取值为百分值时，不是相对于背景图片的尺寸大小来计算，而是相对于元素宽度来计算。例如，这个例子元素 `div` 的宽度是 320px，当 `background-size` 取值为 (50% 80%) 时，此时背景图片的尺寸变成了宽度为 160 ($320 \times 50\%$) px，高度为 216 ($270 \times 80\%$) px。实际效果如图 4-18 所示。

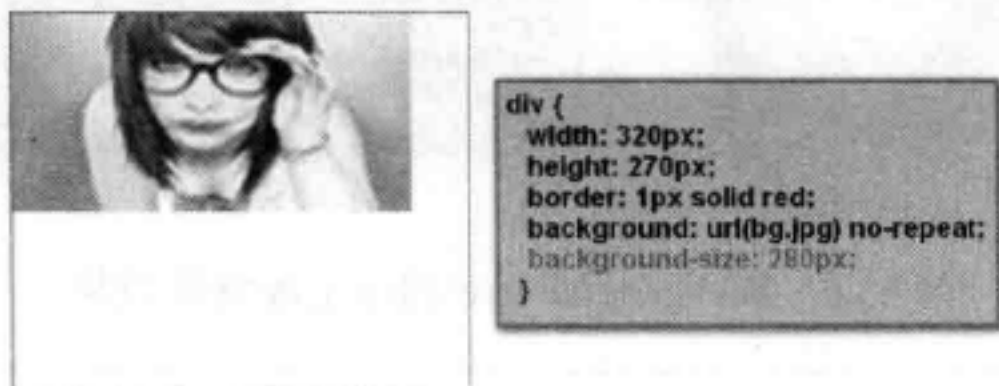


图 4-17 background-size 取一个值的效果

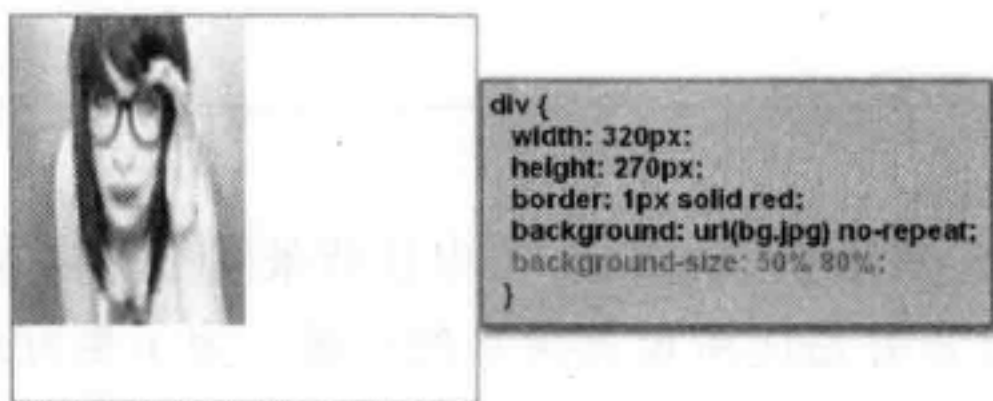


图 4-18 background-size 取值为百分比的效果

`background-size` 取值为百分值时和取值为浮点数字值时一样，都可以只设置一个值，第二个值也是根据图片的比例自动缩放。但有一点需要注意，当元素 `div` 有内距 `padding` 时，还需要加上这个内距值，再进行计算。

3. cover

`background-size` 取 `cover` 值代码如下。

```
div {
    width: 320px;
    height: 270px;
    border: 1px solid red;
    background: url(bg.jpg) no-repeat;
    background-size: cover;
}
```

效果如图 4-19 所示。整个背景图片放大填充了整个元素 `div`。

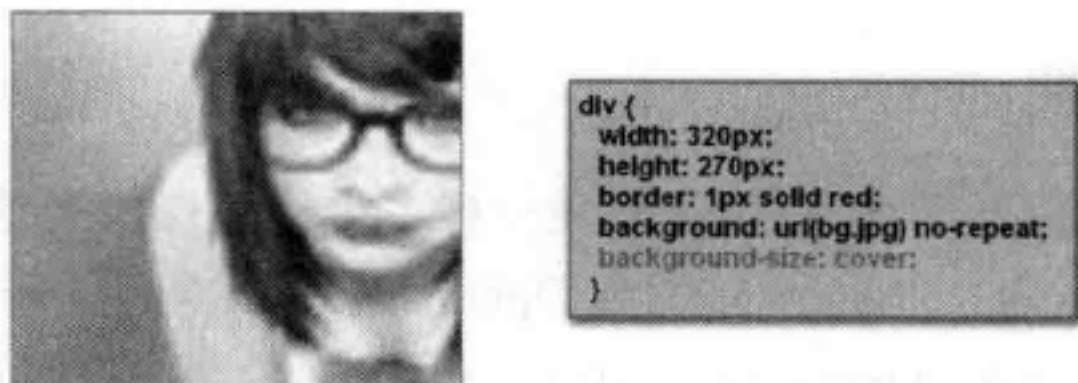


图 4-19 background-size 取值为 cover 的效果

有一个细节需要注意，此时的背景图片放大后致使背景图片不是正中间，为了让背景图片放大后在中间显示，需要在元素中设置 `background-position` 为 `center`，如图 4-20 所示。



图 4-20 `background-size` 制作全屏背景效果



注意

`background-size: cover` 配合 `background-position: center` 常用来制作全屏背景效果。唯一的缺点是，需要制作一张足够大的背景图片，不然在较大分辨率浏览器下会致使背景图片失真。

4. contain

`background-size` 还有一种取值 `contain`，可以让背景图像保持本身的宽高比例，将背景图片缩放到宽度或者高度正好适应所定义背景的区域。为了更好地说明 `contain` 的运行机制，将前面实例中 `div` 大小改成 50px 的宽和高（比背景图片小）写上代码。

```
div {
    width: 50px;
    height: 50px;
    border: 1px solid red;
    background: url(bg.jpg) no-repeat;
    background-size: contain;
}
```



```
div {
    width: 50px;
    height: 50px;
    border: 1px solid red;
    background: url(bg.jpg) no-repeat;
    background-size: contain;
}
```

效果如图 4-21 所示。整个背景图像根据背景区域对背景图像进行了宽高比例的缩放。和 `cover` 值不同，`contain` 在某些情况之下无法让背景图像填充整个容器的大小，而相同之处是在背景图像没处理好时，也会使背景图像失真。






从上面的几个 demo 效果可以看出，只有当 `background-size` 值为默认值 `auto` 时，背景图像才不会失真，而其他值都会造成背景图像失真，所以使用时需要仔细考虑，以免带来不良的效果。

4.4.3 浏览器兼容性

`background-size` 属性和其他的 CSS3 属性一样，得到了现代浏览器的较好支持，在 IE 9+、Firefox 4+、Chrome 4+、Safari 3.1+ 和 Opera 10.5+ 版本浏览器下都可以使用 W3C 的标准语法，但在 Firefox 3.6、Opera 10 ~ 10.1、Safari 3 等老一点版本下同样需要加上各浏

览器供应端的前缀，如表 4-5 所示。

表 4-5 background-size 兼容性

属性名					
background-clip	9 + √	3.6 + √	4.0 + √	10 + √	3 + √

浏览器的兼容性说明，background-size 属性在 IE 8 以及其以下的低版本浏览器是无法正常工作的，而且也很难直接模拟实现这个特性。但在一些实际运用中，如果背景图像只是一个小的视觉效果，使用渐进增加来处理背景图像，仅从这个角度来说，在 IE 下的缺陷就没有必要过于担心。

也可以通过 Modernizr 为 IE 提供一个备选样式，Modernizr 可以检测到浏览器是否支持 background-size 属性。这样可以为不支持 background-size 属性的浏览器提供一个不同的背景图片，这种方案特别适合于背景图像缩放的情景之下。

4.4.4 实战体验：制作全屏背景

background-size 的 cover 可以在不同分辨率的浏览器下使用一张背景图像，也常配合 CSS3 Media Queries Module 一起使用，将达到想不到的效果。也就是说，一张背景图像在不同分辨率的浏览器下都显示全屏效果，具体使用看下面的代码。

```

/* 不支持 CSS3 Media Queries 浏览器按 background-size:cover 模式显示背景图片 */
body {
    background: #000 url(myBackground_1280x960.jpg) center center fixed no-repeat;
    -moz-background-size: cover;
    -webkit-background-size: cover;
    -o-background-size: cover;
    background-size: cover;
}
/*1024px X 768px */
@media only all and (max-width: 1024px) and (max-height: 768px) {
    body {
        -moz-background-size: 1024px 768px;
        -webkit-background-size: 1024px 768px;
        -o-background-size: 1024px 768px;
        background-size: 1024px 768px;
    }
}
/*640px X 480px*/
@media only all and (max-width: 640px) and (max-height: 480px) {
    body {
        -moz-background-size: 640px 480px;
        -webkit-background-size: 640px 480px;
        -o-background-size: 640px 480px;
        background-size: 640px 480px;
    }
}

```

上例中后面两段代码分别实现了 1024 × 768 和 640 × 480 两种分辨率下背景图像显示全屏，加上最前面那段代码在浏览器不支持 CSS3 Media Queries Module 时，背景图像将按 cover 缩放形式显示。

除了上面演示的例子之外，还可以在以下场合使用 background-size 特性。

- ❑ 在流体布局或者响应式布局中，确保背景图像能始终适配容器大小。
- ❑ 对于平铺的重复性背景图像，可以确保背景图像不会有截断效果。
- ❑ 在流体布局中缩放背景图像来伪造出多列分栏效果。
- ❑ 解决 Retina 屏幕双倍像素下背景图像模糊问题。
- ❑ 使用链接或者列表元素的背景图像能和文本一起进行缩放。

4.5 内联元素背景图像平铺循环方式

CSS3 为内联元素添加了一个 background-break 属性，用来指定内联元素背景图像进行平铺时的循环方式，其有三个属性值 bounding-box、each-box、continuous，分别代表三种平铺循环方式。可惜这个属性到目前仅属于 Firefox 浏览，而且其属性要写成“-moz-background-inline-policy”。

background-break 三个属性值分别如下。

- ❑ bounding-box：背景图像在整个内联元素中进行平铺。
- ❑ each-box：背景图像在行内中进行平铺。
- ❑ continuous：下一行的背景图像紧接着上一行中的图像继续平铺。

background-break 属性受限于浏览器的支持力度，目前使用度极低，仅在 Firefox 下能实现，而且还需要修改属性写法，在此仅让大家参考。

4.6 CSS3 多背景属性

CSS3 多背景从其字面上而知，就是设置多个背景图像，这也是 Web 设计师最开心的变化之一。换句话说，就是能够在单一容器上使用复合的背景图像。

在 CSS3 之前，每个容器只能指定一张背景图像，因此每当需要增加一张背景图像的时候，必须至少添加一个容器来容纳它。早期使用嵌套 div 容器显示特定背景的做法不是很复杂，但是它明显难以管理与维护。它让 HTML 标记更加复杂同时也会增加页面文件大小。如果要增加某个图片效果，不仅需要修改 CSS 还需要修改 HTML 代码。

通过 CSS3 的多背景属性，HTML 标记就不需要任何修改，在 CSS 的 background-image 或者 background 属性中列出需要使用的所有背景图像，用逗号分隔开。而且每张图片都具有 background 中的属性，例如可以定位、设置重复、改变背景图像大小以及其他可以单独控制的特性。

话又说回来,要理解 CSS3 的多背景使用,就需要理解各种背景属性的语法和值。无论拥有一个还是多个背景图像,所有背景属性(包括 background-image 和简写的 background 属性)值的语法都是相同的。

4.6.1 CSS3 多背景语法及参数

CSS3 多背景语法和 CSS 中背景语法其实没有本质上的区别,只是在 CSS3 中可以给多个背景图像设置相同或不不同的 background-(position||repeat||clip||size||origin||attachment) 属性。其中最重要的是在 CSS3 多背景中,相邻背景之间必须使用逗号分隔开。具体的语法如下所示。

```
background : [background-image] | [background-position][background-size] |
             [background-repeat] | [background-attachment] | [background-clip] |
             [background-origin],*
```

可以把上面的缩写拆解成以下形式。

```
background-image: url1,url2,...,urlN;
background-repeat: repeat1,repeat2,...,repeatN;
background-position: position1,position2,...,positionN;
background-size: size1,size2,...,sizeN;
background-attachment: attachment1,attachment2,...,attachmentN;
background-clip: clip1,clip2,...,clipN;
background-origin: origin1,origin2,...,originN;
background-color: color;
```

CSS3 多背景的属性参数和 CSS 的背景属性参数类似,只是在其基础上增加了 CSS3 为背景添加的新属性,例如 background-clip、background-origin 和 background-size。在此重温 CSS3 下 background 属性的详细参数吧。

- ❑ background-image: 设置元素的背景图片,可以使用相对地址或绝对地址索引背景图像。
- ❑ background-repeat: 设置元素背景图像的平铺方式,默认值为 repeat。
- ❑ background-position: 设置元素的背景图像定位起点,默认值是 left top。
- ❑ background-size: 设置元素的背景图像的尺寸大小,默认值是 auto。
- ❑ background-attachment: 设置元素的背景图片是否为固定,默认值为 scroll。
- ❑ background-clip: 控制元素的背景图像显示区域大小,默认值为 border-box。
- ❑ background-origin: 控制元素的背景图像 position 的默认起始点,默认值为 padding-box。
- ❑ background-color: 设置元素背景颜色。

除了 background-color 以外,其他的属性都可以设置多个属性值,不过前提是元素有多个背景图像存在。如果这个条件成立,多个属性值之间必须使用逗号分隔开。其中

`background-image` 需要设置多个, 而其他属性可以选择一个或多个, 如果一个元素有多个背景图片, 其他属性只有一个属性值时, 表示所有背景图像应用了相同的属性值。



注意 `background-color` 只能设置一个, 如果设置了多个 `background-color` 将是一种致命的语法错误。

4.6.2 CSS3 多背景的优势

CSS3 多背景属性的出现, 使设计师部分摆脱了对 Photoshop 等绘图工具的依赖。伴随着浏览器的支持力度加强, CSS3 多背景功能会得到广泛的使用。

CSS3 多背景也有层次之分, 按照浏览器中显示时图像叠放的顺序从上往下指定的, 最先声明的背景图片将会居于最上层, 最后指定的背景图片将放在最底层。

只在属性中声明对图片的使用还远远不够, 还要分别告诉浏览器, 对这些背景图像应该如何平铺、如何放置以及各自的大小。要这么做, 只需按前面介绍的方法, 通过指定多个 `background-repeat`、`background-size` 和 `background-position` 属性, 可以单独指定背景图像中某个图像的平铺方式、图像的大小以及位置。






在 CSS2 中实现一个多背景效果, 需要在 HTML 标签中添加多个标签, 也就是有多少张背景图片就需要设置多少个 HTML 标签; 另一种方法是将多张背景图像通过 Photoshop 等绘图工具合成在一张图像上, 这样增加后期修改的难度, 页面控制不灵活。综合而言, 这两种方法都不是最佳方案, 不过这两种方法能兼容所有浏览器, 这也是目前为止得到众多 Web 设计师喜欢使用的原因。

CSS3 的多背景特性只需要一个标签, 设计师省去了合成图像的工作量, 还易于代码维护、后期更新。但其浏览器兼容性不强, 这也造成了众多 Web 设计师不敢尝试使用。

4.6.3 浏览器兼容性

CSS3 多背景在现代浏览器下都得到很好的支持, 只是在 IE8 以及其以下的版本以及 Firefox 和 Opera 旧版本浏览器不支持, 如表 4-6 所示。

表 4-6 CSS3 多背景浏览器兼容性

属性名					
<code>background-clip</code>	9 + √	3.6 + √	10.0 + √	10.6 + √	3.2 + √

CSS3 多背景在各支持的浏览器下都是统一写法, 不需要加自己的前缀, 但如果使用 `background-size`、`background-clip`、`background-origin` 时, 还是需要添加各浏览器的前缀。

在一些效果中, 给元素使用多背景仅仅是起到锦上添花的效果, 对于用户要求不是很高的时候可以忽略, 不考虑其兼容性的处理。

但有些场合, 缺少一些图片总体效果会不尽如人意。例如使用多背景制作一个按钮时候, 设计好的按钮有左中右三个图片切片组合而成, 但是在不支持 CSS3 多背景特性的浏览

器下仅显示中间的部分。这样的场景下使用 CSS3 多背景特性就需要谨慎,因为可选的兼容方案并不多。

对于不支持的浏览器,最简单的方案就是提供一张单一的背景图像。通过 Modernizr 分别定义不同的浏览器。当然可以在属性中重复定义 background-image 属性,但特别要注意,多背景属性要写在单一背景属性的后面,而且还要确保这张单一背景图像确实可用。这是处理兼容 CSS3 多背景特性兼容的常用方案,也是最容易的方案,而且不会对多背景特性造成任何的影响。

相比上一种方案来说,嵌套 HTML 标签来显示多背景更稳妥,更具有扩展性,但工作强度也就相应的增加了,而且又回到了 CSS2 时代。如果决定使用这种方案,必须使用 Modernizr 或者 IE 条件注释来检测浏览器,并相应地做不同的处理。否则,在支持多背景图像的浏览器里面背景就会重复显示。因此这种方案也不能叫做兼容方案,因为嵌套的标签在所有的浏览器上都能正常工作,与其这样,倒不如完全不使用多背景图像技术。

使用伪元素显示附加图片这种方案其实就是嵌套 HTML 标签实现多背景图像的变身。通过“::after”或者“::before”等伪元素生成附加元素来放置背景图片,表面上比直接嵌套 HTML 标签更干净一些,但其实是换汤不换药。这种方案解决了一个问题,随着又产生了新的兼容问题,因为“::after”和“::before”在 IE 8 和现代浏览器中得到了很好的支持,可是在 IE 7 及以下的版本中无法得到支持。这时使用脚本让旧浏览器支持这些伪元素,但还得考虑使支持 CSS3 多背景特性的浏览器忽略这些伪元素。这样兼容就变得无比复杂,无穷无尽。

4.6.4 实战体验:制作花边框

为了巩固学习成果,一起来实战一下,通过现学的 CSS3 多背景技术制作一个花边框。在一个 div 元素中制作一个花边框效果,如图 4-22 所示。

实现图 4-23 的效果,切好五张背景图。



☆图 4-22 CSS3 多背景制作花边框



☆图 4-23 五张背景图切片

对于制作图 4-23 效果，可能大家给出的结构会是这样的。

```
<div class="box-wrp">
  <div class="tl"> 左上角背景图 </div>
  <div class="tr"> 右上角背景图 </div>
  <div class="content"> 我使用了五张背景图片。制作这样的效果 </div>
  <div class="bl"> 左下角背景图 </div>
  <div class="br"> 右下角背景图 </div>
</div>
```

然后分别在各个层上定位，马上达到上面的效果，但是有一点实现起来会非常困难，因为四朵花是在边框下面的，当然用定位是可以实现。如果使用 CSS3 多背景特性，做起来就会轻松多了，下面一起来看看使用 CSS3 多背景特性是如何实现图 4-22 的效果。

首先来看 HTML 标签的变化，由六个 div 简化成一个 div。

```
<div class="demo multipleBg"> 我使用了五张背景图片。制作这样的效果 </div>
```

先给这个 demo 加下面的样式。

```
.demo {
  width: 240px;
  border: 20px solid rgba(104, 104, 142, 0.5);
  border-radius: 10px;
  padding: 80px 60px;
  color: #f36;
  font-size: 25px;
  line-height: 1.5;
  text-align: center;
}
```

接下来是制作效果的关键一步——应用多背景。

```
.multipleBg {
  background: url("bg-tl.png") no-repeat left top,
              url("bg-tr.png") no-repeat right top,
              url("bg-bl.png") no-repeat left bottom,
              url("bg-br.png") no-repeat right bottom,
              url("bg-repeat.png") repeat left top;
  /* 改变背景图片的 position 起始点，四朵花都是 border 边缘处起，而平铺背景是在 padding 内边缘起 */
  -webkit-background-origin: border-box, border-box,
                                border-box, border-box, padding-box;
  -moz-background-origin: border-box, border-box, border-box,
                           border-box, padding-box;
  -o-background-origin: border-box, border-box, border-box,
                         border-box, padding-box;
  background-origin: border-box, border-box, border-box,
                     border-box, padding-box;
  /* 控制背景图片的显示区域，所有背景图片超边 border 外边缘都将被剪切掉 */
  -moz-background-clip: border-box;
  -webkit-background-clip: border-box;
```

```
-o-background-clip: border-box;  
background-clip: border-box;  
}
```

4.7 本章小结

本节首先回顾 CSS 的 background 属性开始，引出 CSS3 为 background 属性新增的属性，依次详细介绍了背景属性的原点 background-origin、背景的裁切 background-clip、背景尺寸 background-size、行内元素背景平铺方式 background-break 等属性的使用，并且介绍了 CSS3 中多背景的使用方法与细节。本章涵盖了 CSS 中有关于 background 的所有属性与运用，就算是一个新手，通过本节的学习，也能把 background 属性融会贯通地掌握。

CSS3 文本

在 Web 页面或者 Web 应用程序中设置文本样式是 CSS 最基本的要求，早期的 CSS 文本功能就是给 Web 页面设置文本的字体、字号、颜色、样式、粗细、间距等。随着 CSS3 的出现，文本功能不仅仅局限于这些基本的运用，它给文本功能添加了一些高级的属性设置，如文本阴影属性 `text-shadow`、文本自动换行属性 `word-break`、长单词与 URL 地址自动换行属性 `word-wrap` 和文本溢出属性 `text-overflow` 等，这也是本章介绍的知识点。

5.1 CSS3 文本简介

设置文本样式是任何一个 Web 页面或者 Web 应用程序必不可少的功能，早期的 CSS 对文本的设置已提供了相当多的功能。先简单了解 CSS 早期的文本功能有哪些，以及其各自的使命。

在 CSS 文本功能上主要分为三大类：字体、颜色和文本，接下分别看看 CSS 文本的各自功能。CSS 字体类型如表 5-1 所示。

表 5-1 CSS 字体类型

属性	功能描述	取值
font-family	定义字体的类型	
font-style	定义字体样式	normal (默认值)、italic (斜体)、oblique (倾斜)
font-weight	定义字体粗细。除了关键值设置外，其还可以设置数字，数字越大表示越粗，常用的是 100 ~ 900，其中 100 最细，900 最粗	normal (默认体)、bold (粗体)、bolder (特粗体)、lighter (细体)

(续)

属性	功能描述	取值
font-size-adjust	定义是否强制对文本使用同一尺寸	
font-stretch	定义是否横向拉伸变形字体	
font-variant	定义字体大小写	normal (默认值)、small-caps (小型的大写字母字体)

以上 6 个文本属性都属于文本功能中的字体类型，其中 font-family 是复合属性中必不可少的属性。此外还有一个复合属性 font，如下所示。

```
font:font-style font-weight/line-height font-family;
```

CSS 文本类型有 11 个属性，如表 5-2 所示。

表 5-2 CSS 文本类型

属性	功能描述	取值
word-spacing	定义词与词之间的间距	normal (默认值)、length (设置词与词之间的距离值，可以是负数)
letter-spacing	定义字符之间的间距	normal (默认值)、length (设置字符与字符之间的间距，可以是负值)
vertical-align	定义文本的垂直对齐方式	baseline (默认值)、sub (上标对齐)、super (下标对齐)、bottom (行框底端对齐)、text-bottom (行内文本底端对齐)、top (顶端对齐)、middle (居中对齐)、百分比数字、长度
text-decoration	定义文本的修饰线	none (默认值)、underline (下划线)、overline (上划线)、line-through (删除线) 和 blink (闪烁线)
text-indent	定义文本首行缩进	length (长度单位) 和百分比
text-align	定义文本水平对齐方式	left (左对齐)、center (水平居中)、right (右对齐)、justify (两端对齐)
line-height	定义文本行高	normal (默认值)、长度值、百分比值、数字
text-transform	定义文本大小写	none (默认值)、uppercase (大写)、lowercase (小写)、capitalize (首字大写)
text-shadow	定义文本阴影效果	
white-space	定义文字之间和文本之间的空白符间距	normal (默认值)、nowrap (空白符合并、换行符忽略)、pre (空白符、换行符保留)、pre-wrap (空白符、换行符保留)、pre-line (空白符合并，换行符保留)
direction	控制文本流入的方向	ltr (默认值)、rtl (文本从右到左流入)、inherit (文本流入方向由继承获得)

除了字体和文本类型之外，还包含一个颜色属性 color，主要用来设置文本的颜色。

5.2 CSS3 文本阴影属性

在 text-shadow 还没有出现之前，主要采用 Photoshop 等绘图工具将其制作成图片或

者使用双内容模拟一个阴影效果，不过这两种方案都让人头痛。现在 CSS3 可以直接使用 `text-shadow` 属性对文本设置阴影效果。这个属性有两个作用，产生阴影和模糊主体。这样无须使用图片就能给文本增加质感。

5.2.1 `text-shadow` 属性的语法及参数

实际上，`text-shadow` 曾经在 CSS2 中出现过，但 CSS2.1 版本中又抛弃了，现在 CSS3 版本又重新捡回来了。这说明 `text-shadow` 属性非常值得 Web 设计师重视。现在很多项目中，CSS3 有很多属性被 Web 设计师使用了，在 CSS3 众多属性中，`text-shadow` 属性是重要属性之一，作为 Web 设计师，很有必要学习并掌握这个属性。

要想掌握 `text-shadow` 属性，首先需要了解其语法。

```
text-shadow: none | <length> none | [<shadow>,* <shadow> 或 none | <color> [,<color>]*
```

也就是：

```
text-shadow:[ 颜色 color] x 轴位移 (x-offset) y 轴位移 (y-offset) 模糊半径 (blur-radius), *
```

`text-shadow` 属性一共包含 4 个属性参数，每个属性参数都具有自己的作用。

- ❑ `color`：阴影颜色，定义绘制阴影时所使用的颜色，这个参数可以放在第一也可以放在最后，是一个可选参数，如果没有显式设置阴影颜色，会使用文本的颜色作为阴影颜色。阴影颜色可以是颜色关键词、十六进制颜色、RGB 颜色、RGBA 透明色等。
- ❑ `x-offset`：X 轴位移，用来指定阴影水平位移量，其值可以是正负值，如果为正值，阴影在对象的右边，反之阴影在对象的左边。
- ❑ `y-offset`：Y 轴位移，用来指定阴影垂直方向偏移量，其值可以是正负值，如果为正值，阴影在对象的底部，反之阴影在对象的顶部。
- ❑ `blur-radius`：阴影模糊半径，可选参数，用来设置阴影的模糊半径，代表阴影向外模糊的模糊范围。这个值越大，阴影向外模糊的范围越大，阴影的边缘就越模糊。不过这个值只能是正值，其值为 0 时，表示阴影不具有模糊效果。






可以使用 `text-shadow` 属性来给文本指定多个阴影，并且针对每个阴影使用不同颜色。指定多个阴影时使用逗号将多个阴影进行分隔。`text-shadow` 多阴影效果按照给定的顺序应用，因此前面的阴影有可能会覆盖后面的，但是它们永远不会覆盖文本本身。

5.2.2 浏览器兼容性

`text-shadow` 属性在 CSS2 中出现，但各大浏览器碍于耗费大量的资源，迟迟未支持，因此在 CSS2.1 中被抛弃，如今在 CSS3 中得到了各大主流浏览器支持，如表 5-3 所示。

`text-shadow` 属性和众多

表 5-3 `text-shadow` 浏览器兼容

属性名					
text-shadow	9 + √	3.5 + √	2.0 + √	9.6 + √	4.0 + √

CSS3 属性一样, 难逃 IE 的兼容问题, 为了解决这一问题, 只好使用滤镜 `filter:shadow` 来处理。`filter:shadow` 滤镜作用与 `dropshadow` 类似, 也能使对象产生阴影效果, 不同的是 `dropshadow` 可以产生渐进效果, 使阴影显得更平滑细腻。

`E {filter:shadow(Color= 颜色值,Direction= 数值,Strength= 数值)}`

其中:

- E 是元素选择器。
- Color 用于设定对象的阴影颜色。
- Direction 用于设定投影的方向, 取值为 0 度, 阴影在文本上面; 45 度, 阴影在文本右上角; 90 度, 阴影在文本右边; 135 度, 阴影在文本右下角; 180 度, 阴影在文本下方; 225 度, 阴影在文本左下方; 270 度, 阴影在文本左边; 315 度, 阴影在文本左上方。
- Strength 就是阴影强度, 类似于 `text-shadow` 的模糊半径。

虽然滤镜能解决 IE 下的文本阴影兼容问题, 但制作出来的文本阴影效果和 `text-shadow` 制作效果相差甚远。

除了使用滤镜之外, 还可以使用 `Modernizr` 或者 IE 条件注释来检测浏览器, 并相应地做不同的处理, 比如使用图片来代替阴影文本。但话说回来, 这样的解决方案并不理想, 受限极大, 给设计师带来很大的劳动强度。

5.2.3 实战体验: 制作立体文本

3D 立体文本特效是通过 `text-shadow` 属性来制作的, 其原理很简单, 就是使用多个 `text-shadow` 的属性值叠加, 模拟 3D 立体文本效果。

通过 X 轴和 Y 轴的偏移量可以控制立体角度, 比如生成九个 3D 立体文本效果, 如图 5-1 所示。

接下来来具体看看使用 `text-shadow` 如何实现这九个不同角度的立体文本效果。

HTML 结构:

```
<div class="text-wrap box1">W3cplus (0 deg)</div>
<div class="text-wrap box2">W3cplus (45 deg)</div>
<div class="text-wrap box3">W3cplus (90 deg)</div>
<div class="text-wrap box4">W3cplus (135 deg)</div>
<div class="text-wrap box5">W3cplus (180 deg)</div>
<div class="text-wrap box6">W3cplus (225 deg)</div>
<div class="text-wrap box7">W3cplus (270 deg)</div>
<div class="text-wrap box8">W3cplus (315 deg)</div>
<div class="text-wrap box9">W3cplus (360 deg)</div>
```

对应的 CSS 代码:

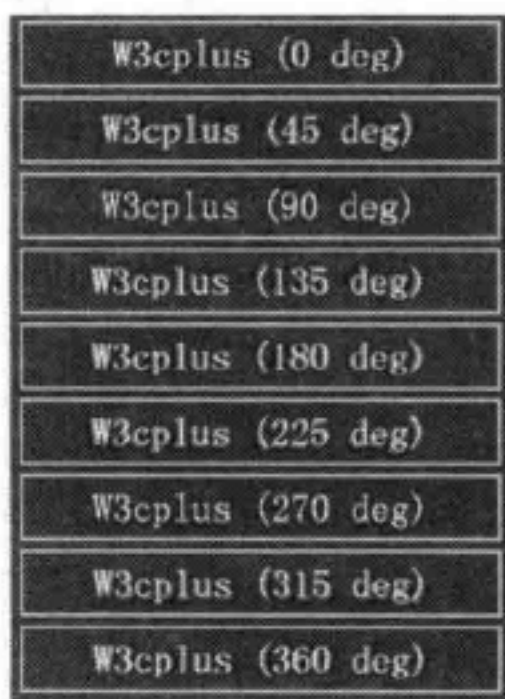


图 5-1 `text-shadow` 制作
立体文本效果


```

<style type="text/css">
body{background-color: #665757;}
.text-wrap {
    width: 600px;
    margin: 10px auto;
    padding: 10px 0;
    border: 5px solid #ccc;
    position: relative;
    box-shadow: 0 0 4px rgba(0, 0, 0, 0.80);
    clear: both;
    font-family: 'Airal', serif;
    font-size: 50px;
    text-align: center;
    color: #f7edf7;
}
/*0° */
.box1 {
    text-shadow: 0px 0px 0 rgb(188,178,188), 1px 0px 0 rgb(173,163,173),
                2px 0px 0 rgb(157,147,157), 3px 0px 0 rgb(142,132,142),
                4px 0px 0 rgb(126,116,126), 5px 0px 0 rgb(111,101,111),
                6px 0px 0 rgb(95,85,95), 7px 0px 0 rgb(79,69,79),
                8px 0px 7px rgba(0,0,0,0.35), 8px 0px 1px rgba(0,0,0,0.5),
                0px 0px 7px rgba(0,0,0,.2);
}
/*45° */
.box2 {
    text-shadow: 0px 0px 0 rgb(188,178,188), 1px -1px 0 rgb(173,163,173),
                2px -2px 0 rgb(157,147,157), 3px -3px 0 rgb(142,132,142),
                4px -4px 0 rgb(126,116,126), 5px -5px 0 rgb(111,101,111),
                6px -6px 0 rgb(95,85,95), 7px -7px 0 rgb(79,69,79),
                8px -8px 7px rgba(0,0,0,0.35), 8px -8px 1px rgba(0,0,0,0.5),
                0px 0px 7px rgba(0,0,0,.2);
}
/*90° */
.box3 {
    text-shadow: 0px 0px 0 rgb(188,178,188), 0px -1px 0 rgb(173,163,173),
                0px -2px 0 rgb(157,147,157), 0px -3px 0 rgb(142,132,142),
                0px -4px 0 rgb(126,116,126), 0px -5px 0 rgb(111,101,111),
                0px -6px 0 rgb(95,85,95), 0px -7px 0 rgb(79,69,79),
                0px -8px 7px rgba(0,0,0,0.35), 0px -8px 1px rgba(0,0,0,0.5),
                0px 0px 7px rgba(0,0,0,.2);
}
/*135° */
.box4 {
    text-shadow: 0px 0px 0 rgb(188,178,188), -1px -1px 0 rgb(173,163,173),
                -2px -2px 0 rgb(157,147,157), -3px -3px 0 rgb(142,132,142),
                -4px -4px 0 rgb(126,116,126), -5px -5px 0 rgb(111,101,111),
                -6px -6px 0 rgb(95,85,95), -7px -7px 0 rgb(79,69,79),
                -8px -8px 7px rgba(0,0,0,0.35), -8px -8px 1px rgba(0,0,0,0.5),
                0px 0px 7px rgba(0,0,0,.2);
}

```

```

}
/*180° */
.box5 {
    text-shadow:0px 0px 0 rgb(188,178,188),-1px 0px 0 rgb(173,163,173),
        -2px 0px 0 rgb(157,147,157),-3px 0px 0 rgb(142,132,142),
        -4px 0px 0 rgb(126,116,126),-5px 0px 0 rgb(111,101,111),
        -6px 0px 0 rgb(95,85,95), -7px 0px 0 rgb(79,69,79),
        -8px 0px 7px rgba(0,0,0,0.35),-8px 0px 1px rgba(0,0,0,0.5),
        0px 0px 7px rgba(0,0,0,.2);
}
/*225° */
.box6 {
    text-shadow:0px 0px 0 rgb(188,178,188),-1px 1px 0 rgb(173,163,173),
        -2px 2px 0 rgb(157,147,157),-3px 3px 0 rgb(142,132,142),
        -4px 4px 0 rgb(126,116,126),-5px 5px 0 rgb(111,101,111),
        -6px 6px 0 rgb(95,85,95), -7px 7px 0 rgb(79,69,79),
        -8px 8px 7px rgba(0,0,0,0.35),-8px 8px 1px rgba(0,0,0,0.5),
        0px 0px 7px rgba(0,0,0,.2);
}
/*270° */
.box7 {
    text-shadow:0px 0px 0 rgb(188,178,188),0px 1px 0 rgb(173,163,173),
        0px 2px 0 rgb(157,147,157),0px 3px 0 rgb(142,132,142),
        0px 4px 0 rgb(126,116,126),0px 5px 0 rgb(111,101,111),
        0px 6px 0 rgb(95,85,95), 0px 7px 0 rgb(79,69,79),
        0px 8px 7px rgba(0,0,0,0.35),0px 8px 1px rgba(0,0,0,0.5),
        0px 0px 7px rgba(0,0,0,.2);
}
/*315° */
.box8 {
    text-shadow:0px 0px 0 rgb(188,178,188),1px 1px 0 rgb(173,163,173),
        2px 2px 0 rgb(157,147,157),3px 3px 0 rgb(142,132,142),
        4px 4px 0 rgb(126,116,126),5px 5px 0 rgb(111,101,111),
        6px 6px 0 rgb(95,85,95), 7px 7px 0 rgb(79,69,79),
        8px 8px 7px rgba(0,0,0,0.35),8px 8px 1px rgba(0,0,0,0.5),
        0px 0px 7px rgba(0,0,0,.2);
}
/*360° */
.box9{
    text-shadow:0px 0px 0 rgb(188,178,188),1px 0px 0 rgb(173,163,173),
        2px 0px 0 rgb(157,147,157),3px 0px 0 rgb(142,132,142),
        4px 0px 0 rgb(126,116,126),5px 0px 0 rgb(111,101,111),
        6px 0px 0 rgb(95,85,95), 7px 0px 0 rgb(79,69,79),
        8px 0px 7px rgba(0,0,0,0.35),8px 0px 1px rgba(0,0,0,0.5),
        0px 0px 7px rgba(0,0,0,.2);
}
</style>

```

上例演示的仅仅是使用 text-shadow 制作的一种文本阴影效果，可以发挥自己的想象力制作出其他的文本效果，比如燃烧字、浮雕字、文本描边等效果。

使用 text-shadow 制作的文本阴影效果，无须修改元素的盒模型属性，因为文本阴影效果不会改变元素的盒子尺寸，但可能会延伸到它的边界之外。同时阴影效果的堆叠层次和元素本身的层次是一样的。

5.3 CSS3 溢出文本属性

平时在网页制作中一定碰到过内容溢出的问题，如文章列表标题很长，而其宽度又受到限制，此时超出宽度的内容就会以省略标记 (…) 显示。以前实现这样的效果都是由后台程序截取一定的字符数在前台输出，另外一种方法就是使用 JavaScript 截取一定的字符数实现。可是这两种方法都有其不足之处，如中文和英文的计算字符宽度的问题，这个值不好计算，所以造成截取字符数不好控制，从而其通用性也差。CSS3 新增了 text-overflow 属性，使得这个问题迎刃而解。

5.3.1 text-overflow 属性的语法及参数

text-overflow 属性解决了以前需要程序或者脚本才能完成的事情，基本语法如下。

```
text-overflow:clip | ellipsis
```

text-overflow 属性参数比较简单，只有两个属性值。






- ❑ clip: 不显示省略标记 (…)，只是简单的裁切。
- ❑ ellipsis: 文本溢出时显示省略标记 (…)，省略标记插入的位置是最后一个字符。

实际上，text-overflow 属性仅用于决定文本溢出时是否显示省略标记 (…)，并不具备样式定义的功能。要实现文本溢出时裁切文本显示省略标记 (…) 效果，还需要两个属性的配合；强制文本在一行显示 (white-space:nowrap) 和溢出内容隐藏 (overflow:hidden)，并且需要定义容器的宽度。

5.3.2 浏览器兼容性

text-overflow 的浏览器兼容性有点特殊，取的属性值不同时，浏览器对其支持也不同，下面分别进行说明，如表 5-4 所示。

表 5-4 text-overflow 的浏览器兼容性

属性名					
text-overflow:clip	6 + √	2.0 + √	1.0 + √	9.63 + √	3.1 + √
text-overflow:ellipsis	6 + √	6.0 + √	1.0 + √	10.5 + √	3.1 + √

text-overflow 属性在 IE 系列下得到较好的支持，直到 Firefox 6 才开始支持 text-overflow:ellipsis 属性的运用，而在 Opera 浏览器下还需要加其独有的前缀 “-o-” 才能识别。

5.3.3 text-overflow 属性使用方法

1. clip

任何理论都需要实践，下面先来看 text-overflow 中 clip 的应用，一起来看下面的一段代码。

HTML 结构：

```
<div class="text-overflow-clip">
  Text-overflow 属性值为 clip 的实战，看看他能不能截取文本？如何截取文本？会怎么显示？
</div>
```

加上 text-overflow:clip 以及一些基本的修饰样式：

```
.text-overflow-clip{
  width: 100px;
  padding: 10px;
  border: 1px solid #ccc;
  text-overflow:clip;
}
```

Text-overflow 属性值为 clip 的实战，看看他能不能截取文本？如何截取文本？会怎么显示？

```
.text-overflow-clip{
  width: 100px;
  padding: 10px;
  border: 1px solid #ccc;
  text-overflow:clip;
}
```

上面示例运行的效果如图 5-2 所示。

图 5-2 text-overflow:clip 的效果

从图 5-2 的效果可以明显看出 text-overflow:clip 没有起到任何作用。文本没有裁切，此时的 div 自动撑高以适应内容的高度。如果把 text-overflow:clip 这句代码删除，都是一样的效果。那需要怎么做呢？前面说过 text-overflow 属性要在一定的高度范围内配合 overflow:hidden 才能生效。现在给这个 div 设置文本不换行和 overflow:hidden 属性。

```
text-overflow-clip{
  width: 100px;
  padding: 10px;
  border: 1px solid #ccc;
  text-overflow:clip; /* 文本裁切 */
  white-space:nowrap; /* 强制不换行 */
  overflow:hidden; /* 溢出隐藏 */
}
```

效果如图 5-3 所示。

此时溢出的文本被截取。也就是说，text-overflow:clip 需要配合 white-space:nowrap（文本强制不换行）和 text-overflow:hidden（文本溢出隐藏）一起使用才能生效，不然它是无任何样式效果。

2. ellipsis

接下来看 text-overflow 取值 ellipsis 属性的使用，开头提到过，text-overflow:ellipsis 在配合 white-space:nowrap、overflow:hidden 属性一起使用可以代替文本截取函数。这样做的最大优点是有助于搜索引擎的搜索。例如，标题有 20 个字符，而文本容器宽度只有 100px，

Text-overflow

```
.text-overflow-clip{
  width: 100px;
  padding: 10px;
  border: 1px solid #ccc;
  text-overflow:clip; /* 文本裁切 */
  white-space:nowrap; /* 强制不换行 */
  overflow:hidden; /* 溢出隐藏 */
}
```

图 5-3 text-overflow:clip 修改后效果

如果采用文本截取函数，文本就显示不够完整，要是换成 `text-overflow:ellipsis` 属性来制作，文本输出是完整的，只不过受到了元素容器大小限制无法全部显示出来，这个时候就用“...”代替隐藏的部分。在前面的实例基础上，修改一下样式代码。

```
.text-overflow-ellipsis{
  width: 100px;
  padding: 10px;
  border: 1px solid #ccc;
  text-overflow:ellipsis;
  white-space:nowrap;
  overflow:hidden;
}
```

效果如图 5-4 所示。

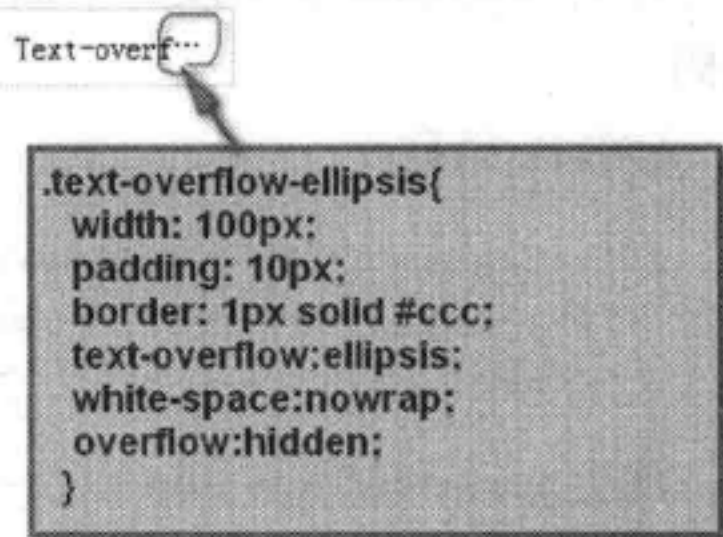


图 5-4 `text-overflow:ellipsis` 效果

综合上面示例，再次证明 `text-overflow` 属性不起任何作用，要让 `text-overflow` 属性起到实际的样式效果，必须同时具备以下三个属性，缺一不可。

- ❑ `width`：明确给需要截取文本的容器设置宽度值。
- ❑ `white-space:nowrap`：给文本容器设置强制不换行，让元素文本一行内显示。
- ❑ `overflow:hidden`：设置容器文本溢出时隐藏。

5.3.4 实战体验：制作固定区域的博客列表^①

博文发布时期紧跟标题后，距离标题 10px。当标题过长时，使用 `text-overflow` 属性将超出部分截断，省略部分使用省略号表示，但是时间必须显示完整，效果如图 5-5 所示。

实现方法很多，不过今天主要现学现卖，使用 `text-overflow:ellipsis` 来制作。

HTML 结构：

```
<div class="block">
  <h3> 最新博文 </h3>
  <ol class="clearfix">
    <li><a href="#"> 如何和何时使用 CSS 的 !important</a>
      <span>2013-02-19</span></li>
    <li><a href="#"> 社交媒体网站设计尺寸参考 </a><span>2013-02-19
      </span></li>
    <li><a href="#">10 个简单有效的方法帮你改善 jQuery 代码与性能 </a>
      <span>2013-01-28</span></li>
    <li><a href="#"> 预处理器的对比——Sass、LESS 和 Stylus</a>
      <span>2013-01-26</span></li>
```



图 5-5 制作固定区域的博客列表

^① 这个案例是迅雷 2009 年页面重构工程师面试题。

```

<li><a href="#">43 个处理触摸事件的 jQuery 插件 </a>
    <span>2013-01-09</span></li>
<li><a href="#">2012 年国外优秀前端网站 </a>
    <span>2013-01-02</span></li>
<li><a href="#">CSS 团队精神: CSS 最佳实践团队开发 </a>
    <span>2012-12-31</span></li>
<li><a href="#">使用 CSS3 的 background-size 优化苹果的 Retina 屏幕的图像显示 </a>
    <span>2012-12-26
    </span></li>
<li><a href="#">你应该知道的一些事情——CSS 权重 </a><span>2012-12-21</span></li>
<li><a href="#">使用 CSS Scriptes 来优化你的网站在 Retina 屏幕下显示 </a>
    <span>2012-12-18</span></li>
</ol>
</div>

```

以上是最普通的列表结构,关键是 CSS 的实现方法,在写样式时先调用一个基本样式文件,过滤浏览器的一些默认样式。

```

<link rel="stylesheet" type="text/css" href=
"http://www.w3cplus.com/demo/css3/base.css" media="all" />

```



注意 这里直接调用了 w3cplus (<http://www.w3cplus.com>) 官网的基本样式文件。

实现效果的主要样式代码如下。

```

<style type="text/css">
    .block {
        margin: 0 auto;
        margin-top: 20px;
        width: 318px; /* 固定博客列表栏目外框 */
        line-height: 20px;
        border: 1px solid #fc9;
    }
    .block ol {
        padding-left: 23px;
        width: 14em; /* 固定标题列表宽度 */
        background: url(bg.png) no-repeat 2px 4px;
    }
    .block li {
        clear: both;
        margin: 0;
        padding: 0;
        list-style: none outside none;
    }
    .block li a {
        float: left;
        _display: inline; /* 兼容 ie6 */
        max-width: 14em; /* 为应用 text-overflow 做准备, 固定宽度 */
        white-space: nowrap; /* 为应用 text-overflow 做准备, 禁止换行 */
    }

```



```

    _white-space:noraml; /* 超长就换行，第二行被裁掉 @ie6 */
    _height: 20px; /* ie6 下隐藏换行的文本 */
    overflow: hidden; /* 为应用 text-overflow 做准备，隐藏溢出文本 */
    text-overflow: ellipsis;
    -o-text-overflow: ellipsis; /* 兼容 opera 浏览器 */
    color: #333;
    _background:transparent; /* 解决莫名占据高度 bug @ie6 */
}
.block li span {
    _position: relative; /* 应付父容器 hLayout 裁切 bug @ie6 */
    float: left;
    _display: inline;
    margin-right: -99px;
    padding-left: 10px;
    font-size: 10px;
    color: #999;
}
</style>

```

5.4 CSS3 文本换行

浏览器自身都带有让文本换行的功能。在浏览器显示文本时，会让文本和浏览器或者文本容器的右端自动实现换行（阿拉伯语等语除外）。对于西方文本来说，浏览器会在半角空格或连字符的地方自动换行，而不会在单词的中间突然换行；对于中文来说，可以在任何一个文字后面换行，但浏览器碰到标点符号时，通常将标点符号以及其前面的一个文字作为一个整体统一换行。

在 text-overflow 的每个示例中都用到了 white-space 属性，用来禁止文本换行。为了增强文本换行显示的功能，CSS3 中又添加了 word-wrap 属性，其属于 IE 的专有属性。

大家在平时的网页制作中一定碰到过这样的情况，在博客中制作一个完美而且又靓丽的评论布局，用户浏览网页时可以添加评论，但当有人发布一个原始网址或者其他超长的文本时，这个博客评论的布局就会被这些长文本彻底的破坏了，为了解决这样的问题平时可能是这样来处理的。

- ❑ 在评论的容器上增加一个 overflow-x:auto 属性设置，当内容超过容器时，在容器底部使用滚动条来控制。
- ❑ 直接使用 overflow:hidden 属性设置，隐藏溢出文本，来达到布局的完美性。
- ❑ 通过 JavaScript 脚本来控制。

虽然以上几种方法都可以实现长文本内容不撑破容器，但在 CSS3 中提供了更好的实现方法，就是使用 word-wrap 属性。

5.4.1 word-wrap 属性

在 CSS3 中，使用 word-wrap 属性实现长单词与 URL 地址的自动换行。

1. word-wrap 的语法及参数

word-wrap 使用方法很简单，其基本语法如下所示。

```
word-wrap:normal | break-word
```

word-wrap 的属性可以使用以下值。

□ normal: 默认值，浏览器只在半角空格或连字符的地方进行换行。

□ break-word: 将内容在边界内换行（不截断英文单词换行）。

简单来看看 word-wrap 使用这两个值的效果有何不同。

2. word-wrap 的使用方法






在实际应用中，有三种情况出现：全是中文、中英文混排、全是英文。

针对上面所说的情况，创建一个实例。

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Word-wrap 的使用 </title>
  <style>
    div {
      float: left;
      width: 150px;
      margin: 10px;
      font-size: 16px;
      font-family: simsun;
      border: 1px solid #ccc;
    }
  </style>
</head>
<body>
  <div>我是测试文本主，全部是中文，没有任何英文，主要是用来测试，看看效果而已。
我是测试文本主，全部是中文，没有任何英文，主要是用来测试作用，看看效果而已。
我是测试文本主，全部是中文，没有任何英文，主要是用来测试作用，看看效果而已。</div>
  <div>W3CPLUS 是一个前端爱好者的家园，W3CPLUS 努力打造最优秀的 web 前端学习的站点。
W3CPLUS 力求原创，以一起学习，一起进步，共同分享为原则。W3CPLUS 站提供了有关于 css
,css3,html,html5,手机移动端的技术文档、DEMO、资源，与前端爱好者一起共勉。</div>
  <div>W3CPLUS is a front end lover's home, W3CPLUS to create the best web
front end study site. W3CPLUS strive to original to study together,
progress together, share for the principle..</div>
  <div>Use the URL:http://www.w3cplus.com/ css/
using-transparency-in-web-design-dos-and-donts.html,
to test. and user the text:ssssssssssssssssssssssssssssssssssssss
stestsssssssssssssssdtestssadasfsatessssssssssssssdstewsdfa and
number:78895482852268123356463345663214524543123464215646321
467436143167431316464313464643134646431464 to test. </div>
</body>
```

在没有显式设置 word-wrap 属性情况下，浏览器取默认值 normal，此时效果如图 5-6 所示。

表 5-5 word-wrap 的浏览器兼容性

属性名					
word-wrap	6+√	3.5+√	1.0+√	10.0+√	3.1+√

5.4.2 word-break 属性

CSS3 中使用 word-break 属性来决定自动换行的处理方法。通过具体的属性设置，不仅可以让浏览器实现半角空格或连字符后面的换行，而且还可以让浏览器实现任意位置的换行。

1. word-break 的语法及参数

word-break 的语法和 word-wrap 的语法类似，也很简单。

```
word-break:normal | break-all | keep-all
```

word-break 属性用于设置或检索对象内文本的字内换行行为，在出现多种语言的情况下尤为有用。其取值简单的说明如下。

- normal：默认值，根据语言自己的规则确定换行方式，中文到边界上的汉字换行，英文从整个单词换行。
- break-all：可以强行截断英文单词，达到词内换行效果。
- keep-all：不允许字断开。如果是中文把前后标点符号内的一个汉字短语整个换行，英文单词整个换行；如果出现某个英文字符长度超过容器边界，后面的部分将撑破容器；如果边框为固定属性，则后面部分无法显示。

2. word-break 的使用方法

通过示例来展示 word-break 取不同值下的差别。

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Word-break 的使用 </title>
  <style>
    div {
      width: 150px;
      margin: 10px;
      font-size: 16px;
      font-family: simsun;
      border: 1px solid #ccc;
    }
  </style>
</head>
<body>
```

```

<div>我是测试文本主, 全部是中文, 没有任何英文, 主要是用来测试作用, 看看效果而已。</div>
<div>W3CPLUS 是一个前端爱好者的家园, W3CPLUS 努力打造最优秀的 web 前端学习的站点。
W3CPLUS 力求原创, 以一起学习, 一起进步, 共同分享为原则。</div>
<div>W3CPLUS is a front end lover's home, W3CPLUS to create the best
webmobile mutual encouragement.</div>
<div>Use the URL:http://www.w3cplus.com/css/
using_transparency_in_web_design_dos_and_donts.html,
to test. and user the text:ssssssssssssssssssssssssssssssssssssssstes
tessssssssssssssdtestsadasfsatessssssssssssssdstewsdafa and
number:788954828522681233564633456632145245431234642156
46321467436143167431316464313464643134646431464 to test. </div>
</body>
</html>

```

word-break 默认效果 (值取为 normal) 如图 5-8 所示。文本换行是按照浏览器的规则换行, 只有碰到长文本会溢出容器外。这个和 word-wrap 取值为 normal 一样效果。

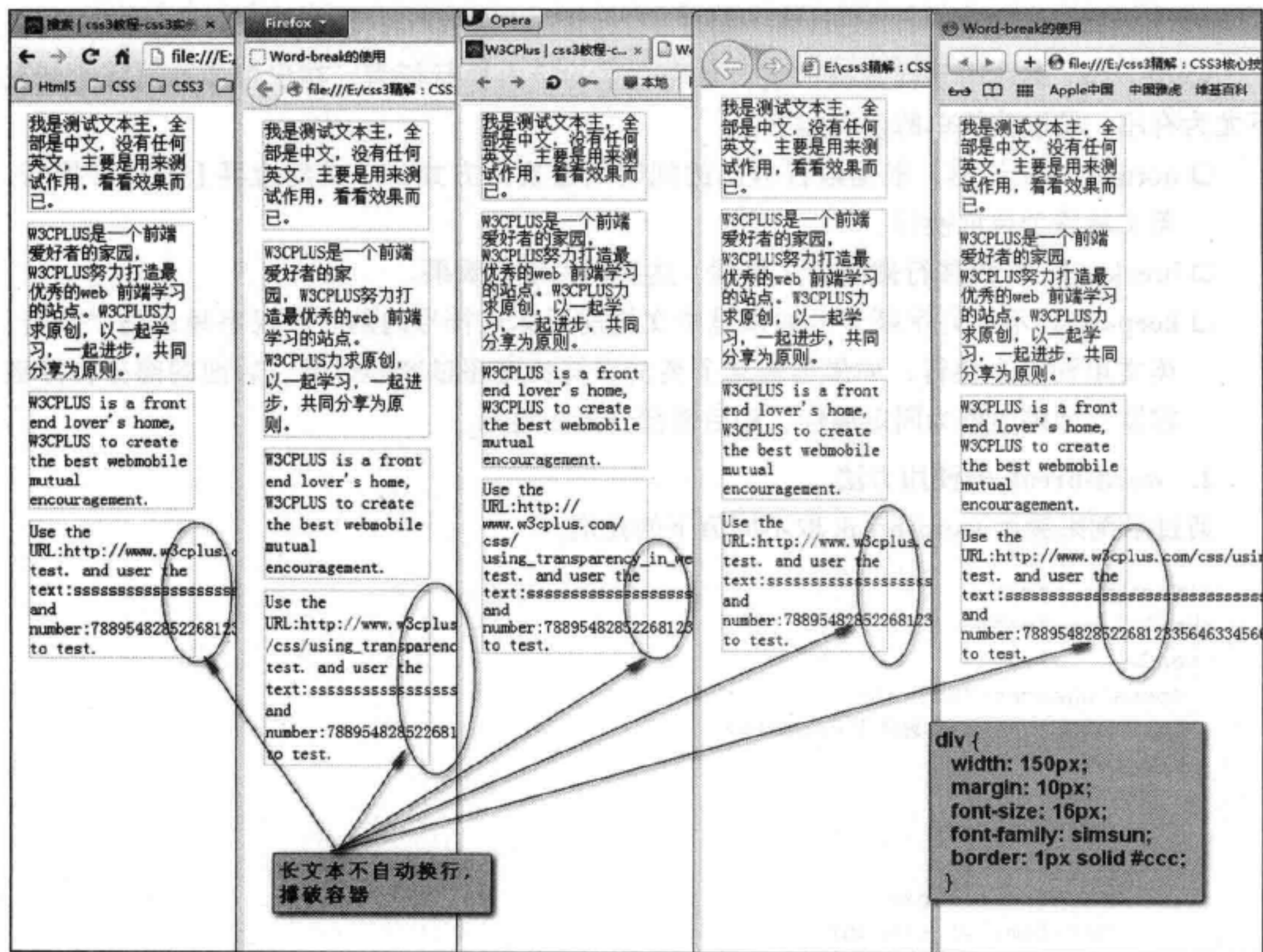


图 5-8 word-break 默认效果

但是 word-break 值设置为 keep-all 所产生的效果在各浏览器下会略有不同。在前面的实例基础中, 在容器 div 上, 显式设置 word-break 属性值为 keep-all。

```
div {
    width: 150px;
    margin: 10px;
    font-size: 16px;
    font-family: simsun;
    border: 1px solid #ccc;
    word-break: keep-all;
}
```

当 word-break 取值为 keep-all 值时, 在 Chrome 和 Safari 浏览器下不起任何效果, 如图 5-9 所示。

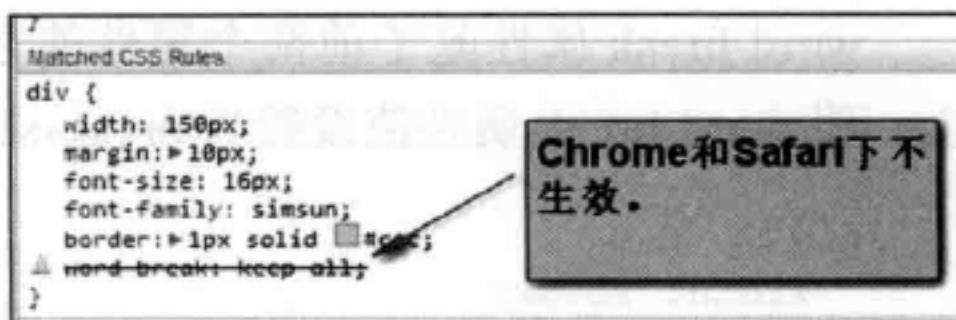


图 5-9 word-break:keep-all 在 Chrome 和 Safari 浏览器下不生效

在 IE 浏览器中, 当 word-break 属性取值为 keep-all 时, 对于中文来说, 只能在半解空格或连字符或任何标点符号的地方换行, 中文与中文之间不能换行, 如图 5-10 所示。



图 5-10 word-break:keep-all 在 IE 浏览器中效果

在 Firefox 浏览器中, 当 word-break 取值为 keep-all 时, 对于中文来说, 只能在半角空格或连字符的地方换行, 中文与中文之间也不能换行, 如图 5-11 所示。

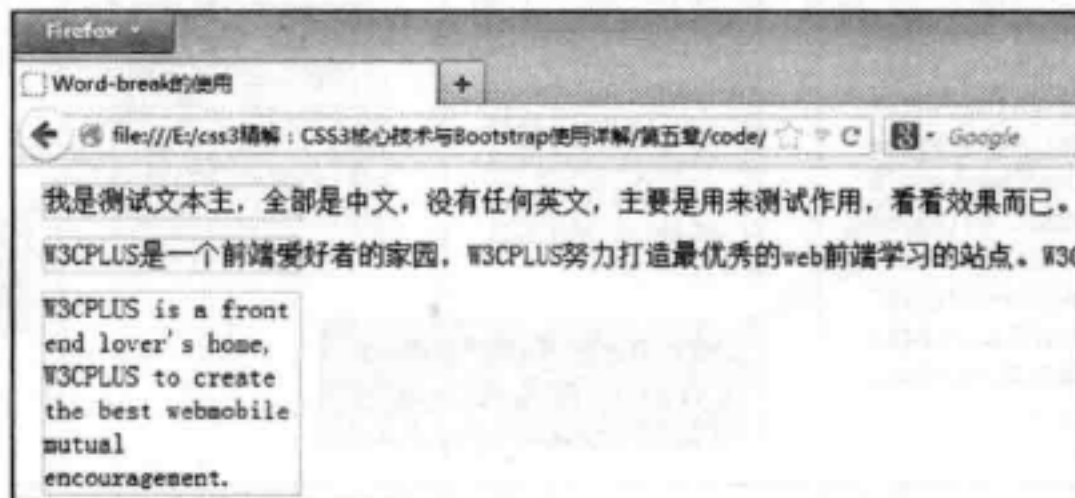


图 5-11 word-break:keep-all 在 Firefox 下效果






word-break 属性取值 break-all, 对于西方文本来说, 允许在单词内换行, 但在 Opera 浏览器中依然无法让长文本 (较长英文单词或 URL 地址) 自动换行。

而对于标点符号来说, 当 word-break 属性使用 break-all 值时, 在 Chrome、Safari 和 Firefox 浏览器中, 允许标点符号位于行首, 但在 IE 浏览器中, 仍然不允许标点符号位于行首。

3. word-break 浏览器兼容性

word-break 属性取值不同时, 浏览器对其支持度也不一样, 如表 5-6 所示。

表 5-6 word-break 的浏览器兼容性

属性名					
word-break:normal	6 + √	3.5 + √	6.0 + √	10.0 + √	3.0 + √
word-break:keep-all	6 + √	3.5 + √	×	10.0 + √	×
word-break:break-all	6 + √	3.5 + √	6.0 + √	×	3.0 + √

5.4.3 white-space 属性

在介绍 text-overflow 属性时, 要起到作用就需要一个关键属性 white-space 的配合。white-space 属性主要用来声明建立布局过程中如何处理元素中的空白符。

1. white-space 的语法及参数

white-space 属性早在 CSS2.1 中就出现了, CSS3 在此基础上增加了两个属性值, 其语法如下。

```
white-space: normal || pre || nowrap || pre-line || pre-wrap || inherit
```

white-space 属性取值简单说明如下。

- normal: 默认值。空白处会被浏览器忽略。可以通过这个值恢复到属性的默认值。
- pre: 文本空白处会被浏览器扣留, 其行为方式类似于 HTML 中的 <pre> 标签效果。
- nowrap: 文本不会换行, 文本会在同一行上, 直到碰到换行标签
 为止。
- pre-line: 合并空白符序列, 但保留换行符, 此属性不支持 IE 7.0-、Firefox 3.0- 和 Opera 9.2- 以下版本浏览器。
- pre-wrap: 保留空白符序列, 但是正常进行换行, 此属性值不支持 IE 7.0 和 Firefox 3.0 以下版本浏览器。
- inherit: 继承父元素的 white-space 属性值, 此属性值在所有的 IE 浏览器都不支持。

2. white-space 的使用方法

上面简单描述了 white-space 属性在不同浏览器取值时的功能。为了能让大家更好地理解这几个值之间的差异和使用效果, 通过几幅效果图来对比。

white-space:normal 用来忽略文本间多余的空白, 但词与词之间的正常空白符会保留在

那里，如图 5-13 所示。

`white-space:pre` 效果和 `<pre>` 展示的效果极其相似，它会保留代码中输入的文本格式，空白符和换行符都将和输入的一模一样，如图 5-14 所示。

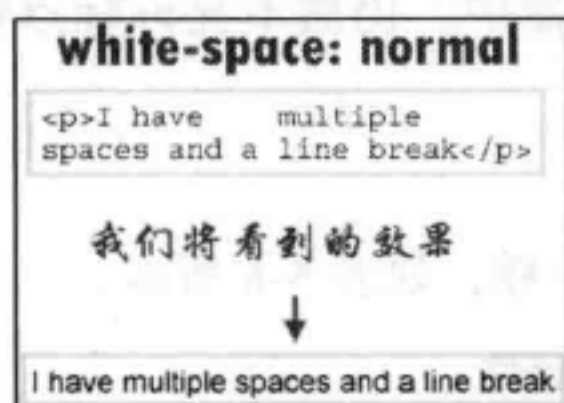


图 5-13 `white-space:normal` 效果



图 5-14 `white-space:pre` 效果

`white-space:nowrap` 常和 `text-overflow` 属性配合使用，它的功能和 `word-break:keep-all` 类似，强制文本不换行，文本会在同一行上显示，直到碰到换行标签 `
` 为止，而且文本中的多余空白文本符会忽略，如图 5-15 所示。

`white-space:pre-line` 具有 `white-space:normal` 的功能，它会将文本之间多余空白文本符忽略，但它还有一个功能，会保留换行符，效果如图 5-16 所示。

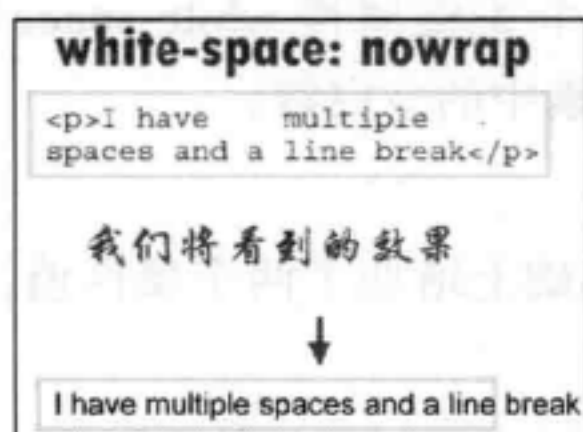


图 5-15 `white-space:nowrap` 效果

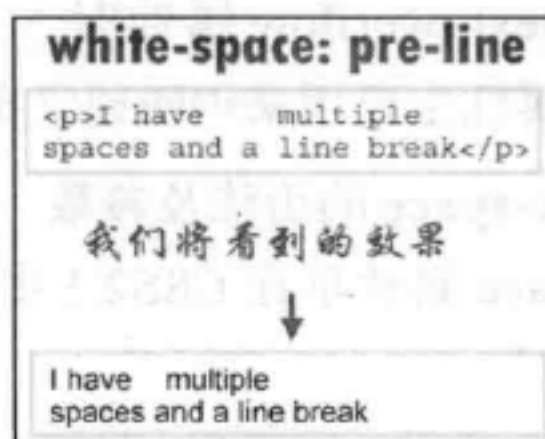


图 5-16 `white-space:pre-line` 效果

`white-space:pre-wrap` 类似于 `white-space:pre` 效果，会保留文本之间的空白符和换行符，如图 5-17 所示。








图 5-17 `white-space:pre-wrap` 效果

3. `white-space` 浏览器兼容性

`white-space` 的浏览器兼容性如表 5-7 所示。

表 5-7 white-space 的浏览器兼容性

属性名					
white-space:normal	6+√	3.0+√	6.0+√	9.0+√	3.0+√
white-space:pre	6+√	3.0+√	6.0+√	9.0+√	3.0+√
white-space:pre-line	7+√	3.0+√	6.0+√	9.0+√	3.0+√
white-space:pre-wrap	7+√	3.0+√	6.0+√	9.0+√	3.0+√

5.4.4 文本换行技巧

文本换行方式都介绍完了，但在不同的地方使用时应该用不同的属性集合，下面是项目中常用的一些文本换行技巧。

1) pre 标签自动换行。

```
pre{
    white-space: pre;           /* CSS 2.0 */
    white-space: pre-wrap;     /* CSS 2.1 */
    white-space: pre-line;     /* CSS 3.0 */
    white-space: -pre-wrap;    /* Opera 4-6 */
    white-space: -o-pre-wrap;  /* Opera 7 */
    white-space: -moz-pre-wrap !important; /* Mozilla */
    white-space: -hp-pre-wrap; /* HP Printers */
    word-wrap: break-word;     /* IE 5+ */
}
```

2) 单元格 (td) 自动换行。

```
table {
    table-layout: fixed;
    width: *** px;
}

table td {
    overflow: hidden;
    word-wrap: break-word;
}
```

3) 除 pre、td 标签外其他标签自动换行。

```
element {
    overflow: hidden;
    word-wrap: break-word;
}
```

4) 标签内容强制不换行。

```
element {
    white-space: nowrap;
    word-break: keep-all;
}
```

5.4.5 文本换行技术对比

前面着重介绍了文本换行的 `word-wrap`、`word-break` 和 `white-space` 三个属性，其实对于文本换行属性，IE 下定义了多个属性，例如 `line-break`、`word-break` 和 `word-wrap`。而其中 `word-wrap` 是 CSS3 中新增的一个属性。

- ❑ `line-break` 属性专门负责控制日文换行，国内 Web 设计师接触较少。
- ❑ `word-wrap` 属性可以控制换行，当其取值 `break-word` 时将强制换行，中英文文本都无任何问题，但是对于长串的英文不起作用。也就是说 `break-word` 属性值是用来断词而不是用来断字符。
- ❑ `word-break` 属性主要针对亚洲语言和非亚洲语言进行控制换行。当属性取值为 `break-all` 时，可允许非亚洲语言文本的任意字符断开；当属性取值为 `keep-all` 时，表示在中文、韩文、日文中是不允许字断开的。
- ❑ `white-space` 属性具有格式化文本作用，当属性取值为 `nowrap` 时，表示文本强制不换行；当取值为 `pre` 时，表示显示预定义文本格式。

在 IE 浏览器下，使用 `word-wrap:break-word` 声明可以确保所有文本正常显示。在 Firefox 浏览器下，长串英文会出现问题。为了解决长串英文问题，一般将 `word-wrap:break-word` 和 `word-break:break-all` 一起使用。但是这样又造成一个新的问题，会导致普通英文语句中的单词断行影响阅读。

综合上述，目前主要问题就是长串英文和英文单词会被断开。为了解决这个问题，可使用 `word-wrap:break-word` 和 `overflow:hidden` 结合，而不是 `word-wrap:break-word` 和 `word-break:break-all` 结合。

5.5 本章小结

本章介绍了 CSS 中对文本样式的设置，随后详细介绍了 CSS3 为文本样式新增的属性，使用 `text-shadow` 属性设置文本阴影效果；使用 `text-overflow` 属性控制文本截取后的效果；文本换行功能，主要介绍了 `word-wrap`、`word-break` 和 `white-space` 三个属性，为文本换行带来的便利以及它们的不足之处。

CSS3 颜色特性

“佛靠金装，人靠衣装”，网页也是如此。随着互联网的迅速发展，一个网页给人们留下的第一印象，既不是它的内容，也不是它的设计，而是整体颜色。

为了能够达到人们的需求，Web 设计师除了需要掌握网站制作的技术之外，还必须能够很好地应用 Web 颜色。换句话说，网站颜色的使用好坏，直接影响网站的生存力。

为了能更好地学习 CSS3 中的颜色特性，先一起来了解一些有关于色彩方面的基础知识。

6.1 网页中的色彩特性

在 Web 网页上，显示器中看到的色彩会随着显示器环境的变化而变化。特别是在 Web 页面这个特殊环境之下，色彩的使用就显得更加困难。首先需要理解 Web 下的特殊环境，在了解色彩原理的基础上逐步掌握 Web 颜色的使用方法，才能制作出令人满意的 Web 页面。

6.1.1 网页色彩的表现原理

我们知道有 256 种 Web 安全颜色，其实这 256 种颜色是指 8 位颜色的表现能力，随着科技的发展，现在颜色不局限于 8 位，16 位色彩的总数是 65536 色，也就是 2 的 16 次方，而新增了 24 位元色彩，也就是 2 的 24 次方，即 16777216 种颜色。32 位色就是 2 的 32 次方的发色数，即 16777216 种颜色，不过它增加了 256 阶颜色的灰度。

32 位色和 16 位色肉眼分辨不出来吗？如果用两台品牌型号都一样的显示器，分别调不

同的颜色，就能看出区别。

而在 Web 页面的设计中，颜色主要运用 16 进制数值的表示方法，为了用 HTML 表现 RGB 颜色，使用十六进制数 0 ~ 255，改为十六进制就是 00 ~ FF，用 RGB 的顺序罗列就成为 HTML 颜色编码。例如，在 HTML 编码中“000000”就是指红色（R）、绿色（G）和蓝色（B）都没有，就是 0 状态，也就是黑色。相反“FFFFFF”就是红色（R）、绿色（G）和蓝色（B）都是 255，也就是白色。

显示器是由一个个像素构成，利用电子束来表现色彩。像素把光的三原色：红色（R）、绿色（G）、蓝色（B）组合成的色彩按照科学原理表现出来。一像素包含 8 位元色彩的信息量，有从 0 ~ 255 的 256 个单元，其中 0 是完全无光状态，255 是最亮状态。

6.1.2 Web 页面的安全色

了解了 Web 页面的特性之后，就会知道网页中的颜色会受到各种不同环境的影响。即使网页使用了非常合理、非常漂亮的配色方案，可实际上每个人浏览的时候看到的效果都略有不同，这样一来你的页面颜色就没有达到预期的效果传达给用户。

要解决这个问题，可以在 Web 页面设计的时候就使用 Web 的安全色。那 Web 安全色是什么呢？接下来简单了解一下这方面的基础知识。

1. Web 页面安全色的产生

不同平台（Mac、PC 等）有不同的调色板，不同的浏览器也有自己的调色板。这就意味着对于一幅图，显示在 Mac 上的 Web 浏览器中的图像，与它在 PC 上相同浏览器中显示的效果可能差别很大。选择特定的颜色时，浏览器会尽量使用本身所用的调色板中最接近的颜色。如果浏览器中没有所选的颜色，就会通过抖动或者混合自身的颜色来尝试重新产生该颜色。

为了解决 Web 调色板的问题，人们一致通过了一组在所有浏览器中都类似的 Web 安全颜色。这些颜色使用了一种颜色模型，在这个模型中，可以使用相应的十六进制值 00、33、66、99、CC 和 FF 来表达三原色（RGB）中的每一种。简单点说，Web 页面的安全色是当红色（R）、绿色（G）和蓝色（B）颜色数字信号值为 0、51、102、153、204 和 255 构成的颜色组合，它一共有 $6 \times 6 \times 6 = 216$ 种颜色（其中彩色为 210 种，非彩色为 6 种）。而 216 种特定的颜色就可以安全地应用于所有 Web 中，而不需要担心颜色在不同的硬件环境、操作系统、浏览器之间的变化。

2. Web 页面安全色的应用

216 种 Web 页面安全色在需要实现高精度的渐变效果或显示真彩图像或照片时会略显不足，但在显示徽标或者二维平面效果图时却是绰绰有余。不过可以看到很多站点利用其他非网页安全色做到了独特的设计风格，所以 Web 设计师并不需要一味地追求使用局限于 216 种 Web 页面的安全色，而是应该更好地搭配使用安全色和非安全色。也就是说网页安

全色与非安全色搭配得恰当,才不会让用户看到的效果与设计制作时相差太远,否则设计的效果与用户在浏览器中看到的效果可能会出现严重的偏色。

6.1.3 色彩模式

如果使用 Photoshop 等绘图软件,对色彩的模式并不会太陌生,如图 6-1 所示。

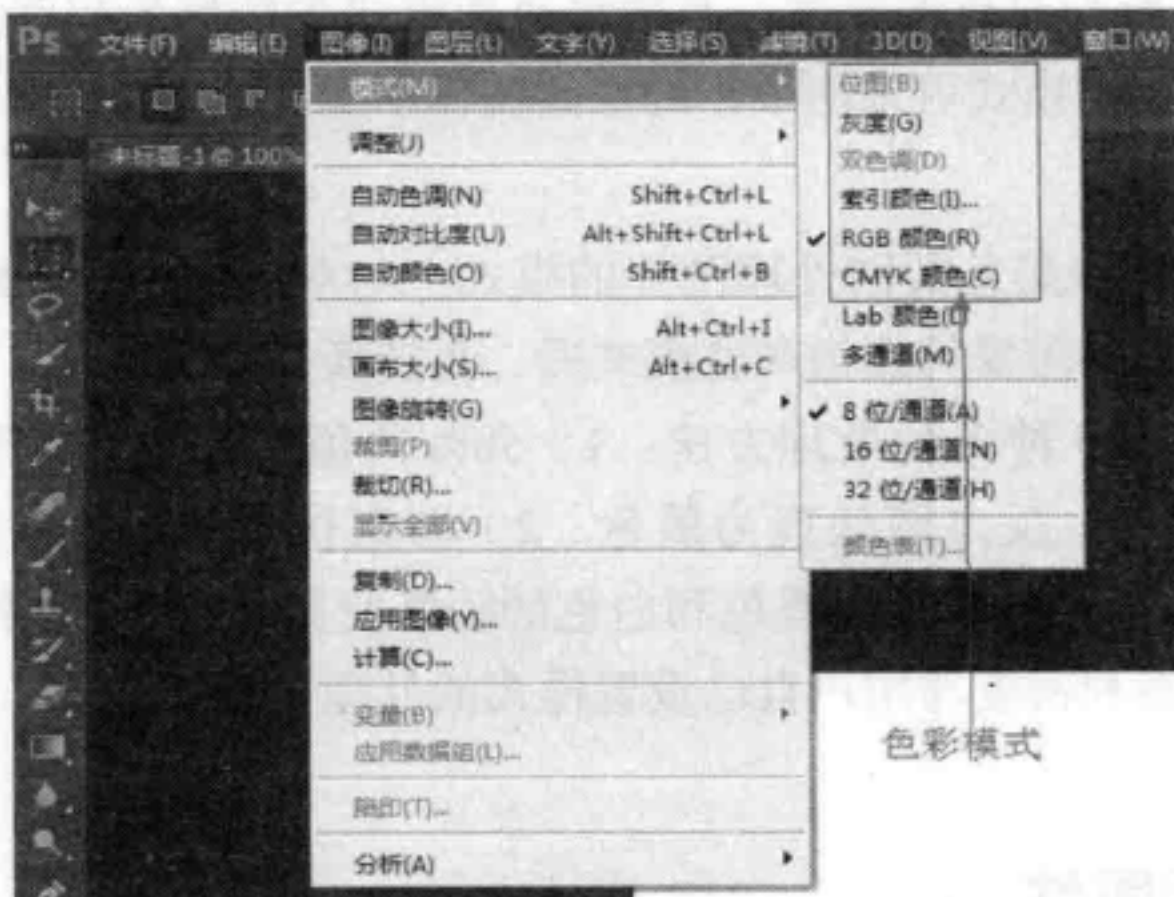


图 6-1 Photoshop 绘图软件的色彩模式

使用 Photoshop 软件制作图片时,首先应制定模式。如果在 Photoshop 中用 CMYK 色彩模式制作图片,为了能够在 Web 页面中使用,需要保存为 JPG 格式。但直接保存是不行的,要把模式转换为 RGB 色彩之后再保存。在 Photoshop 中,RGB 色彩模式的作品如果想转换为双色调模式要怎么做呢?方法是首先转换为灰度模式,然后单击双色调模式的一种就可以了。当然,做好的双色调模式图像要想在网页中使用,还要再次转换为 RGB 色彩模式并进行保存。所以,为了能在计算机图形中使用色彩,就必须了解图像的模式。

1. RGB 色彩模式

RGB 色彩模式是光的三原色红、绿、蓝混合产生的。Web 页面中使用的图片在大多数是在 RGB 色彩模式中制作的。RGB 色彩是颜色相加混合产生的,这样的混合称为加色混合。加色混合中,补色是指相关的两个颜色混合时,成为白色的情况。

2. CMYK 色彩模式

CMYK 色彩模式是指颜料的三原色青色、洋红、黄色加上黑色,这四种颜色减色混合表现出的色彩是主要用于出版印刷时制作图像的一种模式。减色混合是指颜色混合后出现的色彩比原来的颜色暗淡,这样与补色相关的两种颜色混合就会出现彩色的情况。

3. 索引色彩模式

索引色彩模式是已经被限定在 256 种颜色以内的模式,主要用于 Web 页面安全色彩和

制作透明 GIF 图片。在 Photoshop 中制作透明 GIF 图片时，一定要使用索引色彩模式。

4. 灰度模式

灰度模式是无色彩模式，在制作黑白图片时使用，主要用于处理黑、白、灰色图片。

5. 双色调模式

双色调模式是在黑白图片中加入颜色，使色调更加丰富的模式。RGB、CMYK 等颜色模式都不可以直接转换为双色调模式，必须将色彩模式先转换为灰度模式后，才能够转换为双色调模式。用双色调模式可以用很小的空间制作出漂亮的图片。

6. 位图模式

位图模式是用白色和黑色共同处理图片的模式。与双色调一样，除双色调模式和灰度模式外，其他色彩模式都需要转换为灰度模式后，再转换为位图模式。

位图模式可以选定 5 种图片处理方法：1) 50% 阈值，是在 256 种颜色中，当颜色值大于 129 就处理为白色，反之则处理为黑色。2) 图案仿色，是按一定的模式处理图片。3) 扩散仿色为最常用的选项，是按黑色和白色的阴影使其分布。4) 半调网屏与自定图案，是利用盲点的各种形态和密度与用户自己设置样式的处理方式。

6.2 CSS3 透明属性

以前，元素的透明度依靠透明背景图片实现，CSS3 新增加了一个透明属性 `opacity`，可以直接用来设置元素的透明度。

6.2.1 `opacity` 属性的语法及参数

通过 `opacity` 属性能够使任何元素呈现出半透明效果，其语法如下所示。

```
opacity: alphavalue || inherit
```

`opacity` 属性参数非常的简单。

□ `alphavalue`：默认值为 1，可以取 0 ~ 1 任意浮点数。其中取值为 1 时，元素是完全不透明的；反之，取值为 0 时，元素是完全透明不可见。其值不可以是负值。

□ `inherit`：表示继承父元素的 `opacity` 设置的值，即继承父元素的不透明性。

扩展阅读 alpha 通道和 `opacity` 属性的区别

颜色的透明度可以分为 `alpha` 和 `opacity` 两种。`opacity` 是 CSS3 中专门用来设置透明度的一个属性，其取值范围从 0 ~ 1，0 表示完全透明，1 表示不透明。而 `alpha` 通道是用来对元素设置透明度，针对元素的背景色、文字颜色、边框颜色等设置透明度。`opacity` 却只能给整个元素设置一个透明度，并且其透明度直接会继承给其后代元素。

前面说过，`alpha` 通道可以给元素的颜色属性设置透明度，`opacity` 可以用来给整个颜色






设置透明度。除了这两个之外，还可以使用 transparent 属性值给元素的颜色设置完全透明色，例如背景色、边框色、文字颜色、阴影色等，其相当于 alpha 的通道值设置为 0。

在 CSS1 中，只能在 background-color 属性中指定 transparent 值；在 CSS2 中，可以在 background-color 和 border-color 属性中指定 transparent 值；在 CSS3 中，可以在一切指定颜色值的属性中指定 transparent 值。

6.2.2 opacity 浏览器兼容性

除了 IE 8 及其以下版本的浏览器以外，其他主流浏览器都支持 opacity 属性，如表 6-1 所示。

表 6-1 opacity 浏览器兼容性

属性名					
opacity	8+ ✓	1.5+ ✓	1.0+ ✓	9.0+ ✓	3.1+ ✓

虽然 IE 8 以下版本不支持 opacity 透明属性，但可以使用其专有的滤镜来实现 opacity 透明属性的透明效果。

```
/* IE5 - 7 */
filter: alpha(opacity=透明值);
/* IE 8 */
-ms-filter:"progid:DXImageTransform.Microsoft.Alpha(Opacity=透明值);"
```

此处的透明值并不是 0 ~ 1 之间的浮点数，而是 0 ~ 100 之间的任意整数，其中 0 表示元素完全透明不可见。反之，取值为 100 时，元素无任何透明度。

为了兼容 IE 8 以下版本的浏览器，opacity 透明属性在实际中常常如下使用。

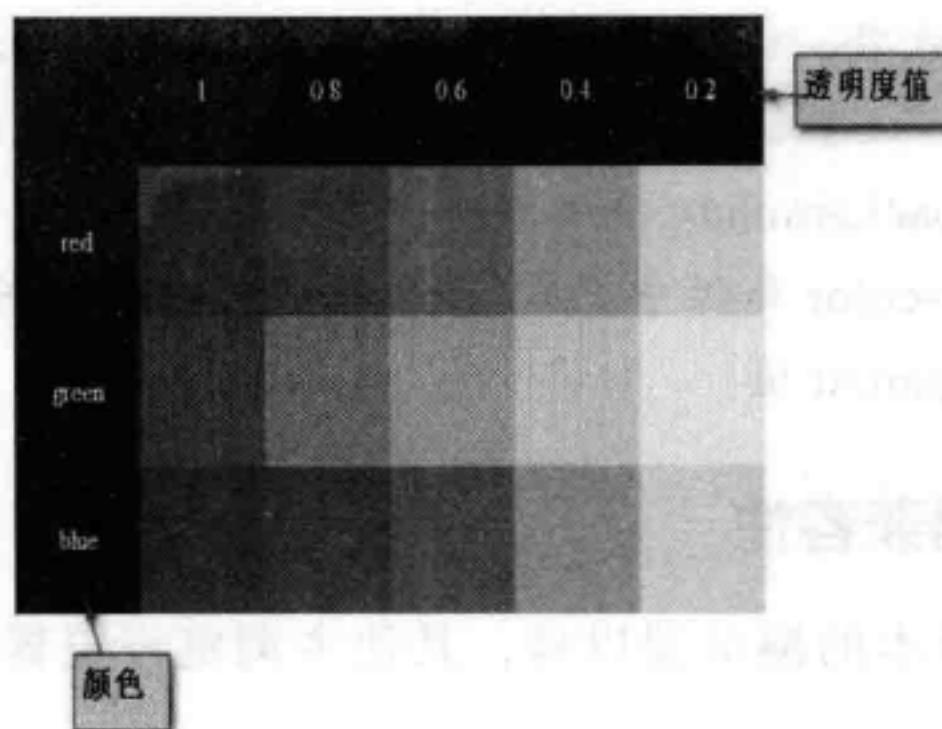
```
/* IE5 - 7 */
filter: alpha(opacity=80);
/* IE 8 */
-ms-filter:"progid:DXImageTransform.Microsoft.Alpha(Opacity=80)";
/* Everyone else */
opacity: 0.8;
```

6.2.3 实战体验：制作透明过渡色块

接下来通过简单的实例演示 opacity 属性在实际中的运用，从而更好地了解 opacity 属性。

例子非常的简单，使用 opacity 属性制作一个透明的过渡色块，效果如图 6-2 所示。

opacity 属性值越小，透明度越大。其最重要的一点是 opacity 属性产生的透明度不仅针对背景色，会针对前景色以及所有后代元素的前景色都会具有一样的透明度效果。实现代码如下。



☆图 6-2 通过 opacity 制作渐变色块 (Chrome 25 下)

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>opacity 制作过渡色块 </title>
  <style>
    .row {
      overflow: hidden;
    }
    .row div {
      width: 80px;
      height: 80px;
      line-height: 80px;
      text-align: center;
      float: left;
    }
    .row:nth-of-type(1) div {
      background: #000;
      color: #fff;
    }
    .row:nth-of-type(2) div {
      background: red;
    }
    .row:nth-of-type(3) div {
      background: green;
    }
    .row:nth-of-type(4) div {
      background: blue;
    }
    .row div:nth-child(1) {
      background: #000;
      color: #fff;
    }
    .row div:nth-child(2) {
      opacity: 1;
    }
  </style>
</head>
</html>
```

```

    }
    .row div:nth-child(3){
        opacity: 0.8;
    }
    .row div:nth-child(4){
        opacity: 0.6;
    }
    .row div:nth-child(5){
        opacity: 0.4;
    }
    .row div:nth-child(6){
        opacity: 0.2;
    }
    .row:nth-of-type(1) div {
        opacity: 1;
    }
</style>
</head>
<body>
    <div class="demo">
        <div class="row">
            <div></div>
            <div>1</div>
            <div>0.8</div>
            <div>0.6</div>
            <div>0.4</div>
            <div>0.2</div>
        </div>
        ...
    </div>
</body>
</html>

```

6.3 CSS3 颜色模式

在 CSS2.1 的基础 CSS3 上新增了 RGBA、HSL 和 HSLA，接下来一起了解这几种颜色模式。

6.3.1 RGBA 颜色模式

在 CSS3 中，RGBA 在 RGB 基础上增加了控制 alpha 透明度的参数，其中 RGB 颜色模式（也称为三原色）是工业界的一种颜色标准，通过对红（R）、绿（G）、蓝（B）三个颜色通道的变化以及它们相互之间的叠加得到各种颜色，RGB 几乎包括人类视力所能感知的所有颜色，是目前运用最广的颜色系统之一。而 RGBA 仅在 RGB 的基础上增加了 alpha 通道，用来设置颜色的透明度。

1. RGBA 的语法及参数

RGBA 在 RGB 的基础上增加了透明通道设置, 其中这个通道值的范围为 0 ~ 1, 其中 0 表示完全透明, 1 表示完全不透明。RGBA 的具体使用方法如下。

```
rgba(r,g,b,a)
```

RGBA 的属性参数很简单, 分别代表红绿蓝以及透明度的值。

□ R: 红色值, 其取值可以是正整数或者百分值。

□ G: 绿色值, 其取值可以是正整数或者百分值。

□ B: 蓝色值, 其取值可以是正整数或者百分值。






□ A: alpha 透明值, 其取值在 0 ~ 1 范围之内。

以上 R、G、B 三个参数, 正整数值的取值范围为 0 ~ 255; 百分数值的取值范围 0 ~ 100%。超出范围的数值将截至其最接近的取值极限。A 参数的取值范围为 0 ~ 1。四个参数值都不可以取负值。

2. 浏览器兼容性

目前为止, RGBA 属性除了 IE 8 及以下的浏览器之外都得到了很好的支持, 如表 6-2 所示。

表 6-2 RGBA 浏览器兼容性

属性名					
RGBA	9+ ✓	3.0+ ✓	1.0+ ✓	10.0+ ✓	3.1+ ✓

3. 实战体验: 制作透明过渡色块

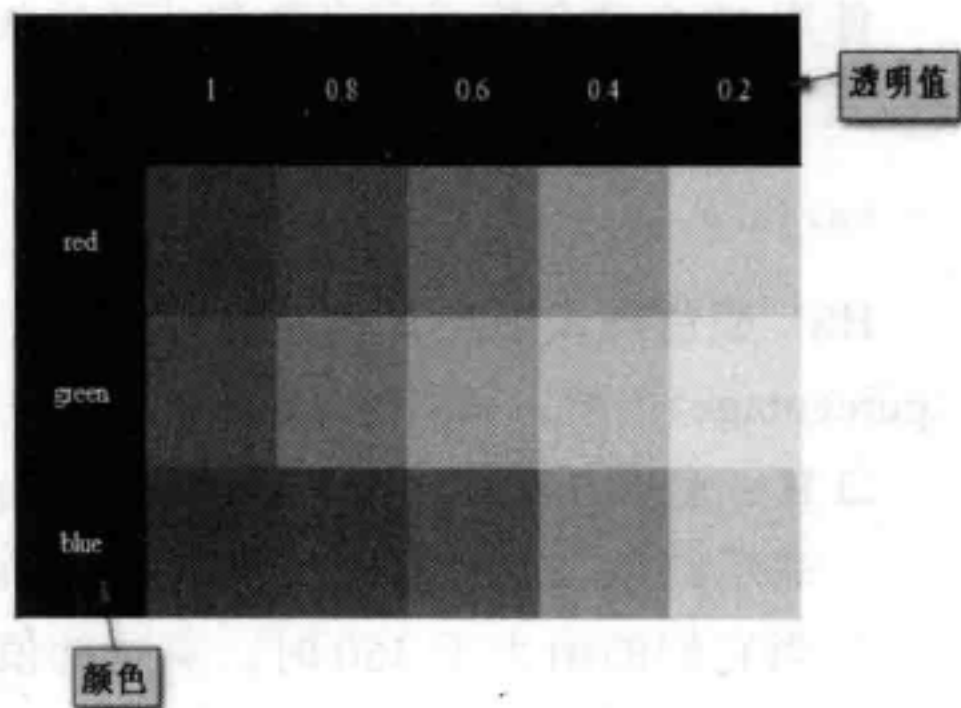
这个案例和 opacity 属性实战的案例一样, 在实例的基础上将 opacity 设置透明度的方法改用 RGBA 来实现。

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>RGBA 制作过渡色块</title>
  <style>
    /* 基本样式代码省略 */
    ...
    /* 红色渐变色块 */
    .row:nth-of-type(2) div:nth-child(3){
      background: rgba(255, 0, 0,0.8);
    }
    .row:nth-of-type(2) div:nth-child(4){
      background: rgba(255, 0, 0,0.6);
    }
    .row:nth-of-type(2) div:nth-child(5){
```

```

        background: rgba(255, 0, 0,0.4);
    }
    .row:nth-of-type(2) div:nth-child(6){
        background: rgba(255, 0, 0,0.2);
    }
    /* 绿色渐变色块 */
    .row:nth-of-type(3) div:nth-child(3){
        background: rgba(0, 128, 0,0.8);
    }
    .row:nth-of-type(3) div:nth-child(4){
        background: rgba(0, 128, 0,0.6);
    }
    .row:nth-of-type(3) div:nth-child(5){
        background: rgba(0, 128, 0,0.4);
    }
    .row:nth-of-type(3) div:nth-child(6){
        background: rgba(0, 128, 0,0.2);
    }
    /* 蓝色渐变色块 */
    .row:nth-of-type(4) div:nth-child(3){
        background: rgba(0, 0, 255,0.8);
    }
    .row:nth-of-type(4) div:nth-child(4){
        background: rgba(0, 0, 255,0.6);
    }
    .row:nth-of-type(4) div:nth-child(5){
        background: rgba(0, 0, 255,0.4);
    }
    .row:nth-of-type(4) div:nth-child(6){
        background: rgba(0, 0, 255,0.2);
    }
</style>
</head>
<body>
    <div class="demo">
        <div class="row">
            <div></div>
            <div>1</div>
            <div>0.8</div>
            <div>0.6</div>
            <div>0.4</div>
            <div>0.2</div>
        </div>
        ...
    </div>
</body>
</html>

```



☆图 6-3 通过 RGBA 制作透明过渡色块

效果如图 6-3 所示。

对比图 6-2 与图 6-3 的效果，opacity 与 RGBA 两个属性都可以轻松实现背景色的透明

效果，虽然效果是一模一样，但是两都之间存在很大的差别。

- `opacity` 属性不仅让背景具有透明度，还使前端色具有相同的透明度，并且其后代元素都将继承其透明度。
- 只是对应属性设置了 `RGBA` 才会有效果，不会影响到元素其他属性或者其他元素的样式风格。
- 区别在于，`opacity` 只能为元素的背景设置透明度（当然会影响元素前景色），但 `RGBA` 可以对元素任何具有颜色的属性设置。换句话说，`RGBA` 除了对元素背景色可以运用之外，还可以运用在 `color` 属性、`border-color` 属性、`box-shadow` 阴影色、`text-shadow` 阴影色等。

6.3.2 HSL 颜色模式

HSL 和 RGB 一样，同属于工业界的一种颜色标准，通过对色调（H）、饱和度（S）、亮度（L）三个颜色通道的变化以及它们相互之间的叠加得到各式各样的颜色的。HSL 标准几乎包括人类视力所能感知的所有颜色，是目前运用最广的颜色系统之一。

使用 HSL 模型为图像中每一个像素的 HSL 分量分配一个 0 ~ 255 范围内的强度值。HSL 图像只用三种通道按照不同的比例混合，在屏幕上呈现 16777216 种颜色。

前面也说过，色调（H）是在色盘上的颜色（如图 6-4 所示），颜色的选择是使用饱和度（S），0 度是红色，120 度为绿色，240 度为蓝色。同时可以使用不同的亮度（L）来控制这个颜色，其中 0 表示的是一个灰度，不使用任何的色彩，而 100% 是指在充分使用一个颜色。

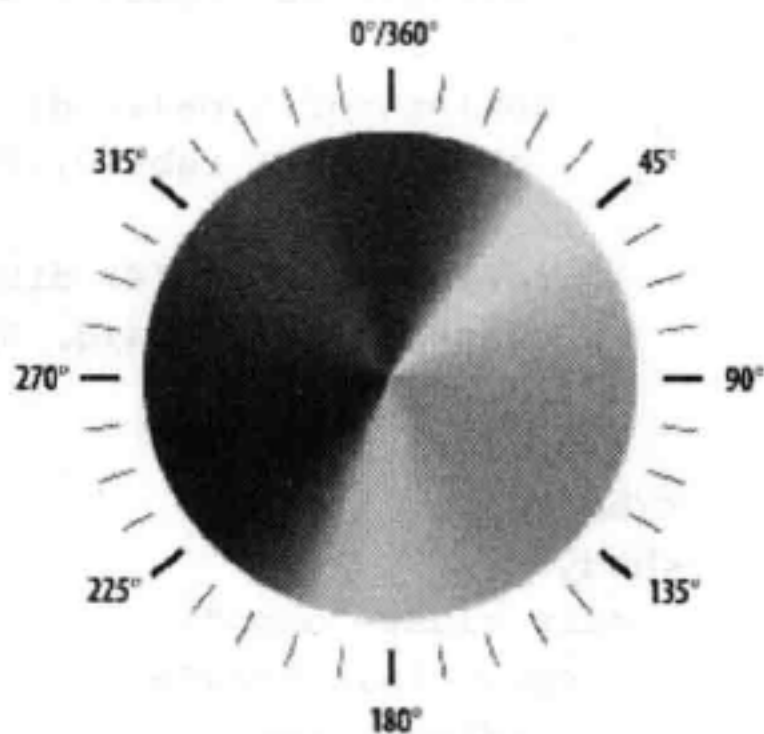
1. HSL 的语法及参数

使用 HSL 颜色模式定义颜色的语法如下，每个值之间使用逗号隔开。

```
hsl(h,s,l)
```

HSL 颜色模式中具有三个属性参数，分别是 `h`（<length>）值、`s`（<percentage>）值和 `l`（<percentage>）值，其具体说明如下。

- **H**：色调（Hue）。取整数值（<length>），可以为任意整数，其中 0（或 360 或 -360）表示红色，60 表示黄色，120 表示绿色，180 表示青色，240 表示蓝色，300 表示洋红。当它们的值大于 360 时，实际的值等于该值除 360 之后的余数。例如，如果色调的值是 480，则实际的颜色值为 480 除以 360 之后得到的余数 120。
- **S**：饱和度（Saturation）。就是颜色的深浅度和鲜艳程序，取百分数（<percentage>），可以取值 0 ~ 100% 之间的任意值，其中 0 表示灰度（没有该颜色），100% 表示饱和



☆图 6-4 HSL 色盘






度最高（该颜色最鲜艳）。

□ L：亮度（Lightness）。取值和饱和度（S）一样，可以取值 0 ~ 100% 之间的任意值，其中 0 最暗（黑色），100% 最亮（白色）。

2. 浏览器兼容性

到目前为止，除了 IE8 及其以下版本浏览器之外，其他主流浏览器对 HSL 色彩模式的支持都比较友好，无须借助各浏览器的私有前缀来实现兼容，兼容检测的结果如表 6-3 所示。

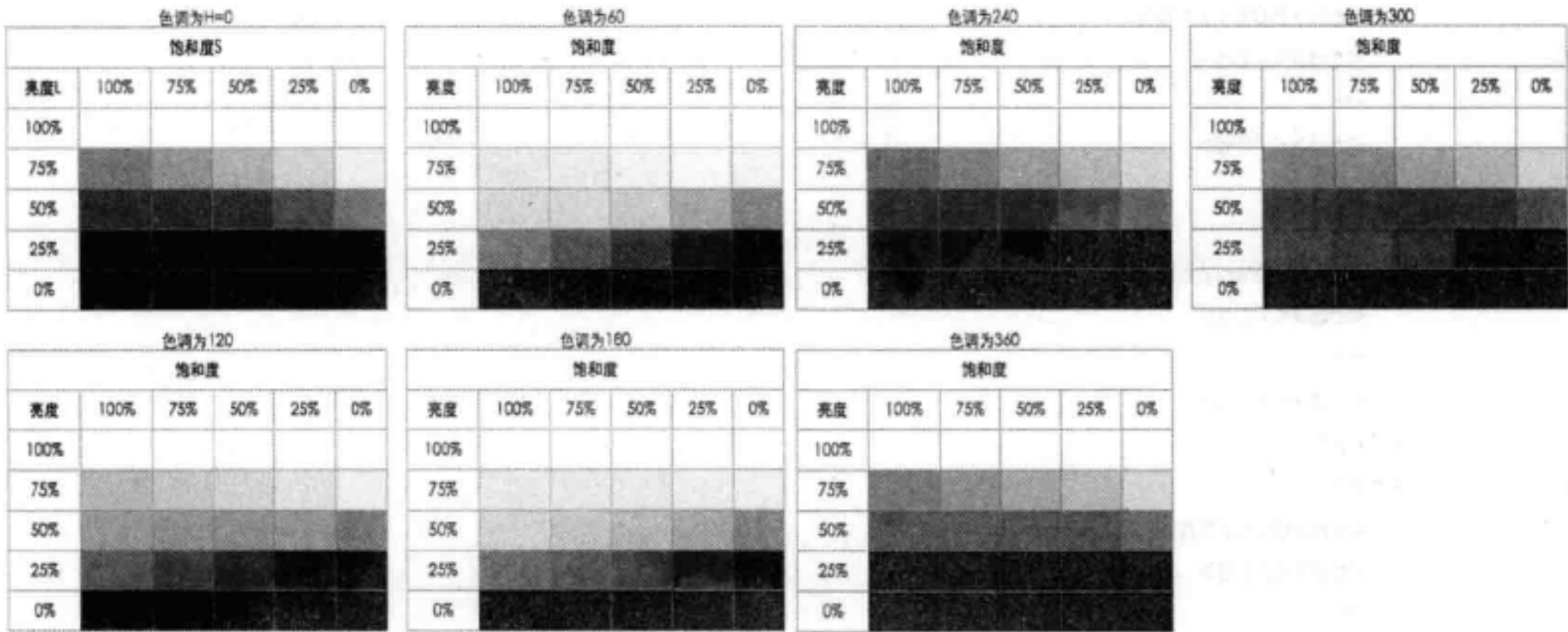
表 6-3 HSL 的浏览器兼容性

属性名					
HSL	9+ ✓	1.5+ ✓	1.0+ ✓	9.6+ ✓	3.1+ ✓

3. 案例实战：制作配色表

在设计一个网站的时候，都是先确定一个主色调，然后在同一个色系中进行选色和配色，这样，在能够保证网页配色丰富，又不会让网页色彩显得花哨。

接下来，一起来看用 HSL 色彩模式制作一个配色模板，模拟的效果如图 6-5 所示。



☆图 6-5 HSL 配色表

由于篇幅所限，仅以颜色色调 H 等于 0 的状态为例，其他状态下的详细代码可以参阅随书代码。示例主要结构使用一个表格来构建。

```
<div class="table-wrap">  
  <table>  
    <caption> 色调为 H=0</caption>  
    <thead>  
      <tr>  
        <th colspan="6"> 饱和度 S</th>  
      </tr>
```

```

    <tr>
      <th> 亮度 L</th>
      <th>100%</th>
      <th>75%</th>
      <th>50%</th>
      <th>25%</th>
      <th>0%</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th>100%</th>
      <td></td>
      ...
      <td></td>
    </tr>
    <tr>
      <th>75%</th>
      <td></td>
      ...
      <td></td>
    </tr>
    <tr>
      <th>50%</th>
      <td></td>
      ...
      <td></td>
    </tr>
    <tr>
      <th>25%</th>
      <td></td>
      ...
      <td></td>
    </tr>
    <tr>
      <th>0%</th>
      <td></td>
      ...
      <td></td>
    </tr>
  </tbody>
</table>
</div>

```

接下来通过前面介绍的伪类选择器，给对应的单元格设置不同的颜色值，而这些颜色值都采用 HSL 颜色模式。下面的代码是 H 值等于 0，而 S 和 L 值从 100% 以 25% 的梯度降至 0 为止，关键代码（详细代码请参阅本书的代码文件）如下所示。

```

/* 色调为 0 */
.table-wrap:nth-child(1) tbody tr:nth-child(1) td:nth-of-type(1) {

```

```

    background: hsl(0,100%,100%);
}
.table-wrap:nth-child(1) tbody tr:nth-child(1) td:nth-of-type(2) {
    background: hsl(0,75%,100%);
}
.table-wrap:nth-child(1) tbody tr:nth-child(1) td:nth-of-type(3) {
    background: hsl(0,50%,100%);
}
.table-wrap:nth-child(1) tbody tr:nth-child(1) td:nth-of-type(4) {
    background: hsl(0,25%,100%);
}
.table-wrap:nth-child(1) tbody tr:nth-child(1) td:nth-of-type(5) {
    background: hsl(0,0%,100%);
}

.table-wrap:nth-child(1) tbody tr:nth-child(2) td:nth-of-type(1) {
    background: hsl(0,100%,75%);
}
.table-wrap:nth-child(1) tbody tr:nth-child(2) td:nth-of-type(2) {
    background: hsl(0,75%,75%);
}
.table-wrap:nth-child(1) tbody tr:nth-child(2) td:nth-of-type(3) {
    background: hsl(0,50%,75%);
}
.table-wrap:nth-child(1) tbody tr:nth-child(2) td:nth-of-type(4) {
    background: hsl(0,25%,75%);
}
.table-wrap:nth-child(1) tbody tr:nth-child(2) td:nth-of-type(5) {
    background: hsl(0,0%,75%);
}

.table-wrap:nth-child(1) tbody tr:nth-child(3) td:nth-of-type(1) {
    background: hsl(0,100%,50%);
}
.table-wrap:nth-child(1) tbody tr:nth-child(3) td:nth-of-type(2) {
    background: hsl(0,75%,50%);
}
.table-wrap:nth-child(1) tbody tr:nth-child(3) td:nth-of-type(3) {
    background: hsl(0,50%,50%);
}
.table-wrap:nth-child(1) tbody tr:nth-child(3) td:nth-of-type(4) {
    background: hsl(0,25%,50%);
}
.table-wrap:nth-child(1) tbody tr:nth-child(3) td:nth-of-type(5) {
    background: hsl(0,0%,50%);
}

.table-wrap:nth-child(1) tbody tr:nth-child(4) td:nth-of-type(1) {
    background: hsl(0,100%,25%);
}
.table-wrap:nth-child(1) tbody tr:nth-child(4) td:nth-of-type(2) {

```



```

    background: hsl(0,75%,25%);
}
.table-wrap:nth-child(1) tbody tr:nth-child(4) td:nth-of-type(3) {
    background: hsl(0,50%,25%);
}
.table-wrap:nth-child(1) tbody tr:nth-child(4) td:nth-of-type(4) {
    background: hsl(0,25%,25%);
}
.table-wrap:nth-child(1) tbody tr:nth-child(4) td:nth-of-type(5) {
    background: hsl(0,0%,25%);
}

.table-wrap:nth-child(1) tbody tr:nth-child(5) td:nth-of-type(1) {
    background: hsl(0,100%,0%);
}
.table-wrap:nth-child(1) tbody tr:nth-child(5) td:nth-of-type(2) {
    background: hsl(0,75%,0%);
}
.table-wrap:nth-child(1) tbody tr:nth-child(5) td:nth-of-type(3) {
    background: hsl(0,50%,0%);
}
.table-wrap:nth-child(1) tbody tr:nth-child(5) td:nth-of-type(4) {
    background: hsl(0,25%,0%);
}
.table-wrap:nth-child(1) tbody tr:nth-child(5) td:nth-of-type(5) {
    background: hsl(0,0%,0%);
}

```

上例中主要演示了色调分别为红色系 0° 和 360° 、黄色系 60° 、绿色系 120° 、青色系 180° 、蓝色系 240° 、洋红色系 300° 为基础系的配色配，当然还可以利用这个色调修改对应的亮度和饱和度制作更多的配色表。

6.3.3 HSLA 颜色模式

HSLA 是 HSL 的扩展模式，在 HSL 的基础上增加一个透明通道 alpha 来设置不透明参数。

1. HSLA 的语法及参数

HSLA 的语法格式如下。






```
hsla(<length>,<percentage>,<percentage>,<opacity>)
```

HSLA 的属性参数与 HSL 的意义和使用方法相同，只是在 HSL 的基础上增加了不透明度 (opacity)。而这个不透明度和 RGBA 颜色模式中的不透明度又是一样的使用方法。其取值也是 $0 \sim 1$ 之间，值越大，透明度越低。

2. 浏览器兼容性

HSLA 颜色模式的浏览器兼容性和 HSL 一样，只有较新的版本的主流浏览器才支持，如表 6-4 所示。

表 6-4 HSLA 的浏览器兼容性

属性名					
HSLA	9+ ✓	3.0+ ✓	1.0+ ✓	10.0+ ✓	3.1+ ✓

3. 实战体验：制作透明过渡色块

这个案例和前面 RGBA 实战的案例极其类似，制作一个透明过渡色块，只不过前者是使用 RGBA 制作透明过渡色块，而此处是使用 HSLA 颜色模块制作的透明过渡色块，大家可以通过这两个案例，进一步理解两者的区别。

```
/*h=0 s=50% l=50%*/
tbody tr:nth-child(1) td:nth-of-type(1) {
    background: hsla(0,50%,50%,1);
}
tbody tr:nth-child(1) td:nth-of-type(2) {
    background: hsla(0,50%,50%,.8);
}
tbody tr:nth-child(1) td:nth-of-type(3) {
    background: hsla(0,50%,50%,.6);
}
tbody tr:nth-child(1) td:nth-of-type(4) {
    background: hsla(0,50%,50%,.4);
}
tbody tr:nth-child(1) td:nth-of-type(5) {
    background: hsla(0,50%,50%,.2);
}
tbody tr:nth-child(1) td:nth-of-type(6) {
    background: hsla(0,50%,50%,0);
}
tbody tr:nth-child(1) td:nth-of-type(1) {
    background: hsla(0,50%,50%,1);
}
tbody tr:nth-child(1) td:nth-of-type(2) {
    background: hsla(0,50%,50%,.8);
}
tbody tr:nth-child(1) td:nth-of-type(3) {
    background: hsla(0,50%,50%,.6);
}
tbody tr:nth-child(1) td:nth-of-type(4) {
    background: hsla(0,50%,50%,.4);
}
tbody tr:nth-child(1) td:nth-of-type(5) {
    background: hsla(0,50%,50%,.2);
}
tbody tr:nth-child(1) td:nth-of-type(6) {
    background: hsla(0,50%,50%,0);
}
```

上面代码仅显示了 $H=0, S=L=50\%$ 状态下，透明值以 0.2 的梯度降从 1 至 0，如图 6-6 所示。其他颜色值示例的详细代码可以参阅书中附带的示例文件。

HSLA制作透明过渡色块						
透明度						
色度	1	0.8	0.6	0.4	0.2	0
0						
60						
120						
180						
240						
300						
360						

☆图 6-6 通过 HSLA 制作的透明过渡色块

6.3.4 RGBA 和 HSLA 颜色模式之间的选择

到目前为止，颜色模式的实际选择中，更多的人喜欢使用 RGBA 颜色模式，主要原因可能是大多数人根本没听说过 HSLA 颜色模式。

使用 RGB/RGBA 颜色值，很难一眼识别出它代表的是什么颜色，需要花一定的时间去研究或者需要借助一定的工具来转出对应的颜色。你很难从红、绿、蓝三个数值中识别出这个颜色的饱和度、亮度是多少。RGB/RGBA 颜色模式的另一个问题是，如果对一个颜色进行调整，就不得不去修改它们的每一项参数。

使用 HSL/HSLA 颜色模式获取一个颜色，没有必要像 RGB/RGBA 颜色模式一样同时调整红、绿、蓝三种颜色，只需要将色调值设置为一个特定值。只记住 0 和 360 代表同一种颜色——纯红色，然后按照彩虹的颜色顺序排列，最终重新回到纯红色，就像图 6-4 所示的色盘。

通过 0 ~ 360 的值确定一个色调后，想将颜色变暗或者变亮之时，只需要调整亮度值（L）。如果想将颜色变浓或者变淡，只需要调整饱和度（S）的值。

它们除了使用方法不同，其他都一样，一个 RGB/RGBA 颜色完全可以使用 HSL/HSLA 来替代。

6.3.5 RGBA/HSLA 的 IE 兼容方案

RGBA 和 HSLA 属性的浏览器兼容性时介绍，IE 8 及以下版本的浏览器下不被支持，在实际使用过程中有一定的方案解决。

- 在 IE 8 及以下版本，一般在前面设置一个其他颜色模式（不具备透明色），其后紧跟一个 RGBA/HSLA 颜色模式。这样，不支持 RGBA/HSLA 的浏览器会以其他的颜色

模式显示 (如 RGB/HSL 等), 而在支持 RGBA/HSLA 的浏览器中会以 RGBA/HSLA 颜色模式显示。

- ❑ 将透明 PNG 图片平铺作为背景图片 (这种方案仅适应于 RGBA/HSLA 运用 background-color 的情况下)。这种方案的优势在于可以真正实现半透明背景。适用于 IE 7 ~ 8, 但不兼容 IE 6 及以下版本, 因为它们不支持透明的背景图片。这种解决方案会增加 HTTP 的请求。相比而言, 如果不考虑兼容 IE 6 及以下版本, 不建议使用 PNG 透明背景图片。
- ❑ 使用 Gradient 滤镜可以指定半透明的颜色。将起止色设置为同一种颜色即可避免产生渐变。

推荐使用第一种和第三种方案, 其中第三种方案的实现结果更接近对视觉效果的要求, 因为背景是半透明的。然而, Gradient 滤镜会对元素文字的抗锯齿产生一些影响。

6.3.6 RGBA/HSLA 滤镜格式

为了在 IE 8 及以下版本中使用 RGBA/HSLA 颜色模式相同的透明度, 需要将 RGBA/HSLA 中的透明值 (如透明度为 0.6) 乘以 255, 然后将其转换成十六进制值。对于 RGBA 转换成滤镜, 可以使用林小志写的转换工具^①, 而对于 RGBA/HSLA 转换成 IE 滤镜可以使用 Michael Bester 编写的工具^②。这些工具可以将 RGBA/HSLA 自动转换成对应的 Gradient 滤镜值。

Gradient 滤镜的颜色值和使用的六位数标准的十六进制代码有所不同, 前两位表示 Alpha 透明度值, 可以使用 00 ~ FF 的任意十六进制值, 其中 00 代表全透明, 而 FF 代表完全不透明。因此, 0.6 的透明度也就是 ($0.6 \times 255 = 153$), 将其转换成十六进制, 就是 99, 而 A6DADC 就是一个颜色的十六进制色。

因此, 在 IE 8 及其以下使用 #A6DADC 并且透明度为 0.6 的透明色时, 转成 Gradient 滤镜值为:

```
-ms-filter: filter:progid:DXImageTransform.Microsoft.gradient(enabled='true',startColorstr='#99A6DADC',endColorstr='#99A6DADC');
```

6.4 本章小结

本章主要介绍了 Web 的颜色模式, 详细介绍了颜色在 Web 中的使用。网页中的颜色特性中主要介绍了色彩特性以及其表现原理, 并且介绍了 Web 页面的安全颜色。色彩模式中详细介绍了 RGB、CMYK、索引色、灰度色、双色调以及位图等色彩模式的各种定义。最后, 介绍了透明度在 Web 中的运用。

① 详见 http://www.linxz.de/css_tool/hex_color.html。

② 详见 <http://kimili.com/journal/rgba-hsla-css-generator-for-internet-explorer>。

CSS3 盒模型

CSS 有一种基础设计模式叫盒模型，定义了 Web 页面中的元素是如何看做盒子来解析的。每一个盒子有不同的展示界面，本章主要介绍一些盒模型以及这盒模型的用户界面是如何展示的。

7.1 CSS 盒模型简介

在 CSS 中主要有以下几种盒模型：inline、inline-block、block、table、absolute position、float。浏览器把每个元素看做一个盒模型，每一个盒模型是由以下几个属性组合所决定的：display、position、float、width、height、margin、padding 和 border 等，不同类型的盒模型会产生不同的布局。

7.1.1 什么是盒模型

CSS 中每一个元素都是一个盒模型，包括 HTML 和 body 标签元素。在平时设计中，大家对 content、padding、margin、background 和 border 来说一定不会陌生了，因为盒模型都具备这些属性。其中 width、height、border、background、padding 和 margin 之间的层次关系和相互影响，看网上的 3D 示意图，如图 7-1 所示。

从图 7-1 中可以看出 padding、content、background-

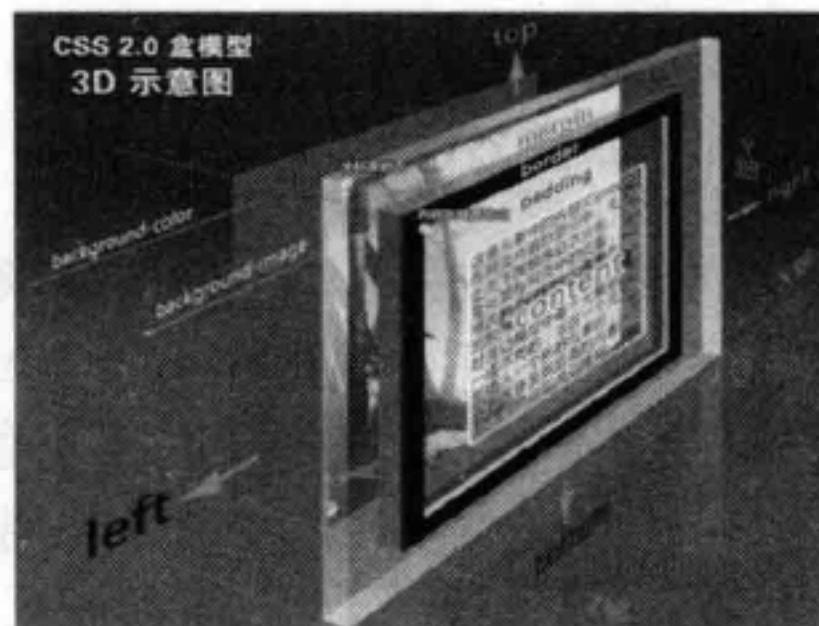


图 7-1 CSS2.0 盒模型 3D 示意图

image、background-color，它们四者构成了Z轴（垂直屏幕的坐标）多重层叠关系。但是border与margin、padding三者之间应该是平面上的并级关系，并不能构成Z轴的层叠关系。

7.1.2 重置盒模型解析模式

在W3C的传统CSS2.1盒模型中，通过声明width和height值来控制内容区域的宽度和高度，然后附加上内边距和边框等，这通常称为内容盒模型。

在CSS中盒模型被分为两种，第一种是W3C的标准模型，另一种是IE的传统模型，它们相同之处都是对元素计算尺寸的模型，具体说就是对元素的width、height、padding和border以及元素实际尺寸的计算关系，不同之处是两者的计算方法不一致。

1) W3C 的标准盒模型。

外盒尺寸计算（元素空间尺寸）

element 空间高度 = 内容高度 + 内距 + 边框 + 外距

element 空间宽度 = 内容宽度 + 内距 + 边框 + 外距

内盒尺寸计算（元素大小）

element 高度 = 内容高度 + 内距 + 边框（height为内容高度）

element 宽度 = 内容宽度 + 内距 + 边框（width为内容宽度）

2) IE 传统下盒模型（IE 6 以下，不包含 IE 6 版本或 QuirksMode 下 IE 5.5+）。

外盒尺寸计算（元素空间尺寸）

element 空间高度 = 内容高度 + 外距（height包含了元素内容宽度、边框、内距）

element 空间宽度 = 内容宽度 + 外距（width包含了元素内容宽度、边框、内距）

内盒尺寸计算（元素大小）

element 高度 = 内容高度（height包含了元素内容宽度、边框、内距）

element 宽度 = 内容宽度（width包含了元素内容宽度、边框、内距）

原则上说盒模型是分得很细的，这里看到的主要是外盒模型和内盒模型，如上面计算公式所示。下面看一个简单的例子，有一个div元素box具有下面一个属性。

```
.box {
  border: 20px solid;
  padding: 30px;
  margin: 30px;
  background: #ffc;
  width: 300px;
}
```

先来看W3C标准浏览器（Firefox、Safari、Chrome、Opera和IE 6+）和传统浏览器（IE 6以下版本浏览器）的layout截图。如图7-2所示，IE 6以下版本浏览器的宽度包含了元素的padding和border值。

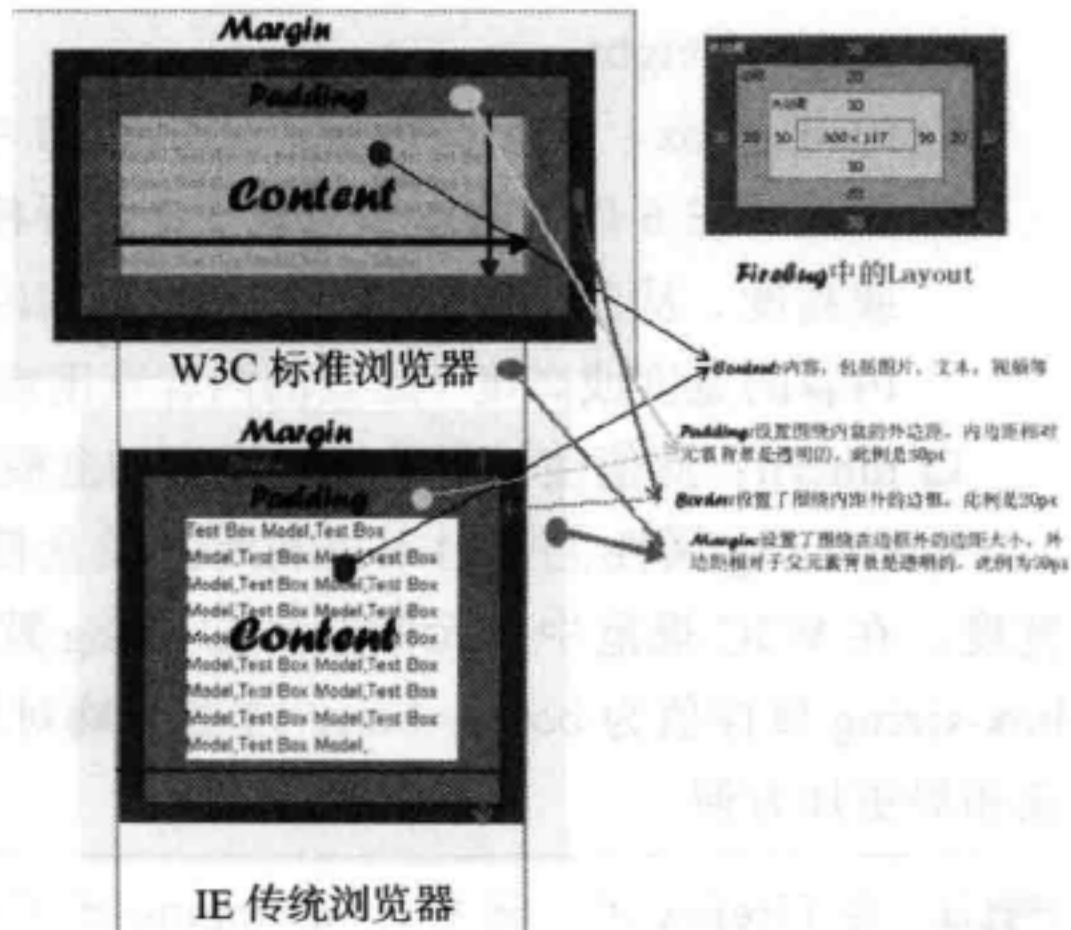


图 7-2 现代浏览器和传统浏览器 layout 截图

换句话说, IE 6 以下版本其内容真正的宽度是 width、padding 和 border。用内外盒来说, W3C 标准浏览器的内容宽度等于 IE 6 以下版本浏览器的外盒宽度。这样就需要在 IE 6 以下的版本写 Hack 统一其外盒的宽度, 关于如何处理这样的兼容问题这里不多说。浏览器发展到今天, 市面上用 IE 6 以下的浏览器应该相当少(可以说已绝迹), 但不排除没有人在使用。所以目前浏览器大部分元素都是基于 W3C 标准的盒模型, 但对于 form 中部分元素还基于传统的盒模型, 例如 input 中的 submit、reset、button 和 select 等元素, 这样如果给其设置 border 和 padding 它们也只会往元素盒内延伸。

7.2 CSS3 盒模型属性

前面阐述了在 IE 5.x 以及 IE 6 ~ 7 的怪异 (Quirks) 模式下, 边框和内距都包含在宽度或高度内。而在标准的浏览器中, 宽度和高度仅仅包含了内容宽度, 除去了边框和内距两个区域, 这样为 Web 设计师处理效果增添了很多的麻烦。

7.2.1 box-sizing 属性的语法及参数

为了解决这种问题, CSS3 增添了一个盒模型属性 box-sizing, 能够事先定义盒模型的尺寸解析方式, 语法如下。

```
box-sizing: content-box | border-box | inherit
```

box-sizing 的属性值主要有以下三个。

- ❑ content-box: 默认值, 让元素维持 W3C 的标准盒模型。元素的宽度和高度 (width/height) 等于元素边框宽度 (border) 加上元素内距 (padding) 加上元素内容宽度或高度 (content width/height), 也就是 $\text{element width/height} = \text{border} + \text{padding} + \text{content width/height}$ 。
- ❑ border-box: 此值会重新定义 CSS2.1 中盒模型组成的模式, 让元素维持 IE 传统的盒模型 (IE 6 以下版本和 IE6 ~ 7 怪异模式)。元素的宽度或高度等于元素内容的宽度或高度。从盒模型介绍可知, 这里的内容宽度或高度包含了元素的 border、padding、内容的宽度或高度 (此处的内容宽度或高度 = 盒子的宽度或高度 - 边框 - 内距)。
- ❑ inherit: 此值使元素继承父元素的盒模型模式。

box-sizing 属性主要用来控制元素的盒模型的解析模式, 其主要目的是控制元素的总宽度。在 W3C 规范中, 元素的 box-sizing 默认属性值是 content-box 值, 如果不显式重置 box-sizing 属性值为 border-box, 才能明确对元素设置一个总宽度。在这种情况下会使页面布局更加方便。



注意 在 Firefox 浏览器中, box-sizing 还可以设置一个 padding-box 属性值, 用来指定元素的宽度或高度包括内容宽度或高度和内距, 但不包括边框宽度。

IE 6 以下版本以及怪异模式下的 IE 浏览器对盒模型虽然不符合 W3C 的标准规范,但这种解析方式并不是一无是处,它也有好的一方面:不管如何修改元素的边框或者内距大小,都不会影响元素盒子的总尺寸,也就不会打乱页面的整体布局。而在标准浏览器下,按照 W3C 规范对盒模型的解析,一旦修改了元素的边框或者内距,就会影响元素的盒子尺寸,也就不得不重新计算元素的盒子尺寸,从而影响到整个页面的布局。

为了更形象地看出 box-sizing 中 content-box 和 border-box 两者的区别,先简单看一下图 7-3 中 box-sizing 中两个属性对盒子模型的不同解析。

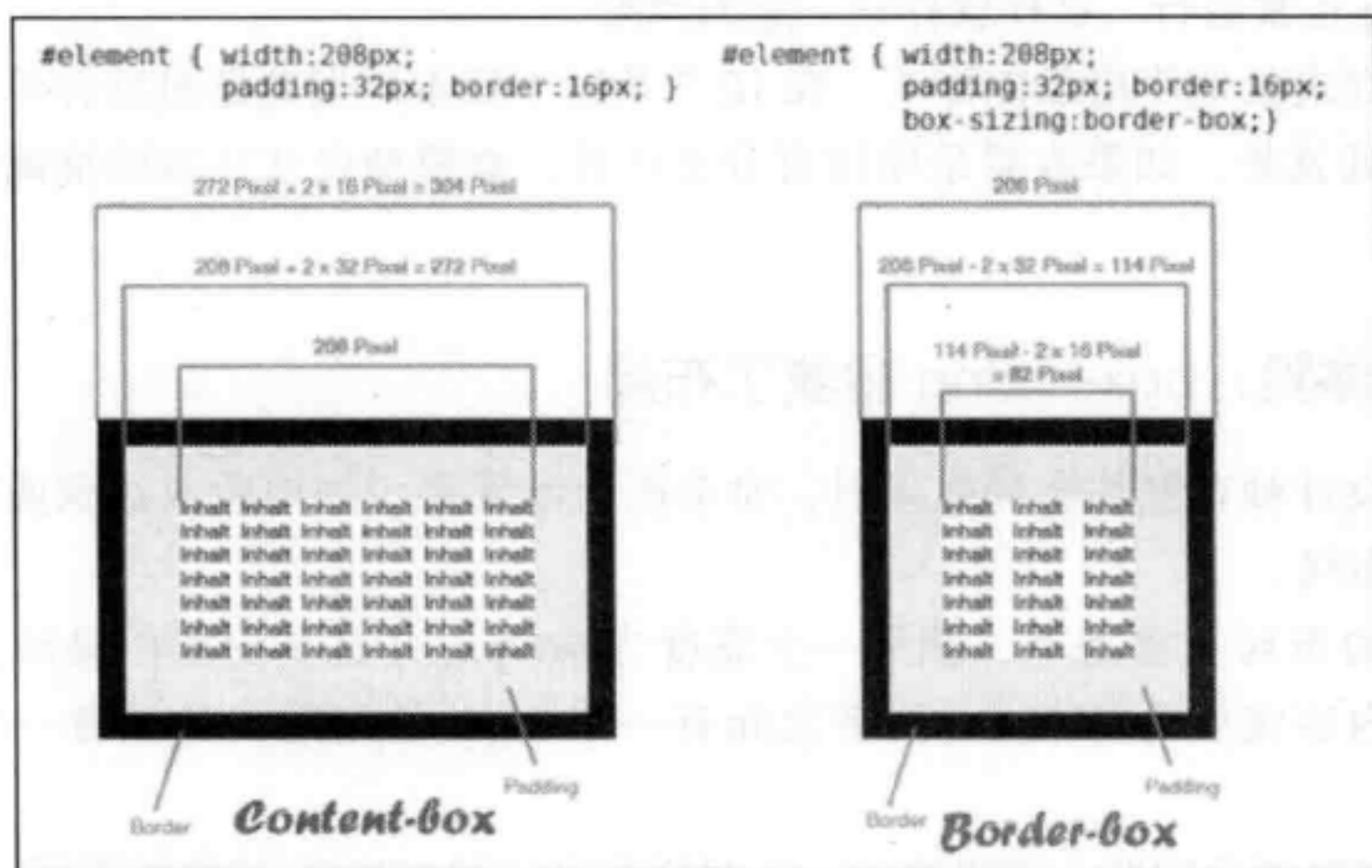


图 7-3 box-sizing 中 content-box 和 border-box 解析模式的对比








注意 图 7-3 示意图来自于 W3C 官网。

7.2.2 浏览器兼容性

目前,只有 Mozilla Gecko 引擎的浏览器支持 box-sizing 属性需要添加其私有属性,其他浏览器都直接支持 box-sizing 属性。各主流浏览器对 box-sizing 属性的支持情况如表 7-1 所示。

表 7-1 box-sizing 浏览器兼容性

属性名					
box-sizing	8+ ✓	1.5+ ✓	1.0+ ✓	9.0+ ✓	3.1+ ✓

在 IE 7 及以下版本的浏览器是不支持 box-sizing 属性的。而 box-sizing 和纯粹装饰性效果的属性不同,因此目前阶段很有必要为 IE 7 及以下版本的浏览器提供一定的替代方案。

让 IE 7 及以下的版本浏览器进入怪异模式,在这个模式下 IE 会使用类似 border-box 盒

模型的解析模式来计算盒模型的尺寸。这并不是一个很理想的解决方案，因为要让 IE 进入怪异模式本身就是一件很难的事情。通常它会要求使用一个非法的 DTD 文档类型，这样一来对其他浏览器会造成问题。另一方面，IE 一进入怪异模式，整个网站的容器都将使用 border-box 的模式来解析每个元素的盒模型，这样就会造成一些正常工作的容器变得不正常，从而需要做更多的修改，得不偿失。

另一种兼容处理方式就是使用 JavaScript 脚本 ie7.js，能够让 box-sizing 属性在 IE7 及以下版本上得到正常使用。这种方案的不足之处是，禁用脚本后整个功能都将失去，无法保证 box-sizing 正常运行。这样就存在一定的风险。

还有一种方式就是使用条件样式，在 IE 7 及以下版本的浏览器重新计算容器的盒模型尺寸。不足之处就是，如果容器是使用百分比计算，盒模型尺寸计算的准确性存在的问题。

7.2.3 实战体验：box-sizing 拯救了布局

Web 前端设计师在制作布局效果时，常会因为给元素添加内距或边框而打破容器，破


```

padding: 0;
}
.wrapper {
width: 960px;
margin-left: auto;
margin-right: auto;
color: #fff;
font-size: 30px;
text-align: center;
}
#header {
height: 100px;
background: #38382e;
margin-bottom: 10px;
}
.sidebar {
float: left;
width: 220px;
margin-right: 20px;
margin-bottom: 10px;
height: 300px;
background: #5d33cf;
}
.content {
float: left;
width: 720px;
height: 300px;
background: #c8ca30;
margin-bottom: 10px;
}
#footer {
background: #cc4ad5;
height: 100px;
clear: both;
}
</style>

```

目前的布局一点问题都没有，因为 wrapper 容器子元素宽度间距加起来刚好与容器 wrapper 相等，但有时候整个布局都将会被打破，例如给 wrapper 容器中的每个子元素设置内距 10px，代码如下。

```

*{
margin: 0;
padding: 0;
}
.wrapper {
width: 960px;
margin-left: auto;
margin-right: auto;
color: #fff;

```

```

    font-size: 30px;
    text-align: center;
    background: #ccc;
}
#header {
    height: 100px;
    background: #38382e;
    margin-bottom: 10px;
    padding: 10px;
    width: 100%;
}
.sidebar {
    float: left;
    width: 220px;
    margin-right: 20px;
    margin-bottom: 10px;
    height: 300px;
    background: #5d33cf;
    padding: 10px;
}
.content {
    float: left;
    width: 720px;
    height: 300px;
    background: #c8ca30;
    margin-bottom: 10px;
    padding: 10px;
}
#footer {
    background: #cc4ad5;
    height: 100px;
    text-align: center;
    clear: both;
    padding: 10px;
    width: 100%;
}

```

在 `#header`、`.sidebar`、`.content` 和 `#footer` 元素上添加内距 10px 时，整个宽度就发生了变化，页眉和页脚总宽度变为 $100\% + 20\text{px}$ ，左边栏 `.sidebar` 变为 $240\text{px}(220 + 20)$ ，主内容变为 $740\text{px}(720 + 20)$ 。如此一来边栏和主内容总宽度为 980px ，如果加上间距就变为 1000px ，远远超过容器 `wrapper` 的宽度，因此整个布局将打破，如图 7-5 所示。页眉和页脚撑破容器伸出去了，主内容也被掉到边栏的下面了。跟想要的效果完全不一样。

此时，有人或许会问，如果不使用内距

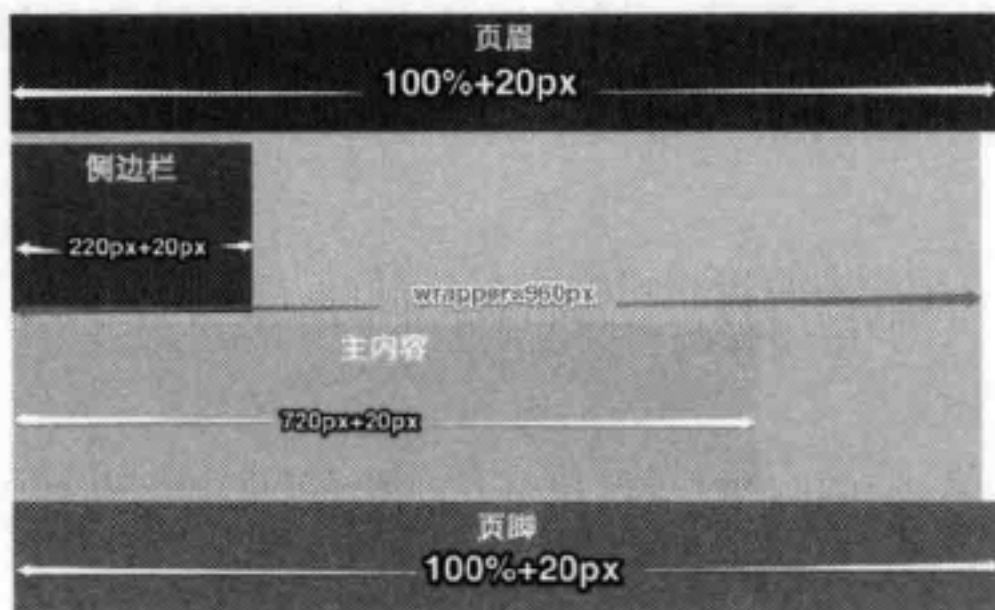


图 7-5 内距打破布局的效果

padding, 只使用边框 border 会怎么样呢? 把刚才的 padding 换成 border, 如下所示。

```
*{
    margin: 0;
    padding: 0;
}
.wrapper {
    width: 960px;
    margin-left: auto;
    margin-right: auto;
    color: #fff;
    font-size: 30px;
    text-align: center;
    background: #ccc;
}
#header {
    height: 100px;
    background: #38382e;
    margin-bottom: 10px;
    border: 10px solid red;
    width: 100%;
}
.sidebar {
    float: left;
    width: 220px;
    margin-right: 20px;
    margin-bottom: 10px;
    height: 300px;
    background: #5d33cf;
    border: 10px solid red;
}
.content {
    float: left;
    width: 720px;
    height: 300px;
    background: #c8ca30;
    margin-bottom: 10px;
    border: 10px solid red;
}
#footer {
    background: #cc4ad5;
    height: 100px;
    text-align: center;
    clear: both;
    border: 10px solid red;
    width: 100%;
}
```

此时依然打破整个布局, 如图 7-6 所示。

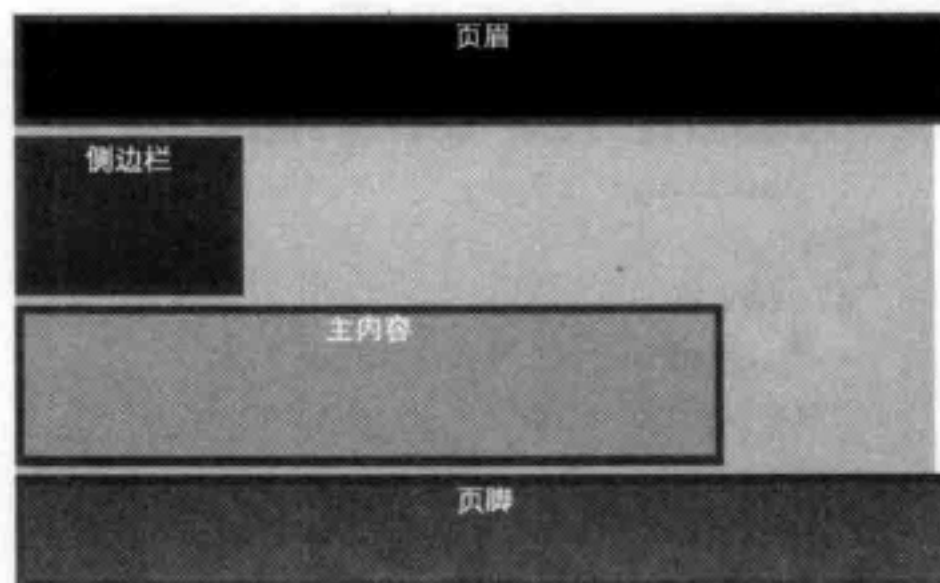


图 7-6 边框打破布局的效果

图 7-6 是去掉 padding 只加 10px 边框, 同样把布局给打乱。接着把 padding 和 border

同时加进去。

```

*{
    margin: 0;
    padding: 0;
}
.wrapper {
    width: 960px;
    margin-left: auto;
    margin-right: auto;
    color: #fff;
    font-size: 30px;
    text-align: center;
    background: #ccc;
}
#header {
    height: 100px;
    background: #38382e;
    margin-bottom: 10px;
    border: 10px solid red;
    padding: 10px;
    width: 100%;
}
.sidebar {
    float: left;
    width: 220px;
    margin-right: 20px;
    margin-bottom: 10px;
    height: 300px;
    background: #5d33cf;
    border: 10px solid red;
    padding: 10px;
}
.content {
    float: left;
    width: 720px;
    height: 300px;
    background: #c8ca30;
    margin-bottom: 10px;
    border: 10px solid red;
    padding: 10px;
}
#footer {
    background: #cc4ad5;
    height: 100px;
    text-align: center;
    clear: both;
    border: 10px solid red;
    padding: 10px;
    width: 100%;
}

```

因为加上 padding 和 border 把布局给打乱了, 如图 7-7 所示。

前面介绍了, 图中 box-sizing 取其默认值 content-box, 其盒模型完全符合 W3C 标准, 为了修复这个布局, 需要把盒模型改用 IE 传统下的解析。

```
*{
  margin: 0;
  padding: 0;
}
.wrapper {
  width: 960px;
  margin-left: auto;
  margin-right: auto;
  color: #fff;
  font-size: 30px;
  text-align: center;
  background: #ccc;
}
#header {
  height: 100px;
  background: #38382e;
  margin-bottom: 10px;
  border: 10px solid red;
  padding: 10px;
  width: 100%;
  -moz-box-sizing: border-box;
  -webkit-box-sizing: border-box;
  -o-box-sizing: border-box;
  -ms-box-sizing: border-box;
  box-sizing: border-box;
}
.sidebar {
  float: left;
  width: 220px;
  margin-right: 20px;
  margin-bottom: 10px;
  height: 300px;
  background: #5d33cf;
  border: 10px solid red;
  padding: 10px;
  -moz-box-sizing: border-box;
  -webkit-box-sizing: border-box;
  -o-box-sizing: border-box;
  -ms-box-sizing: border-box;
  box-sizing: border-box;
}
.content {
  float: left;
  width: 720px;
  height: 300px;
```

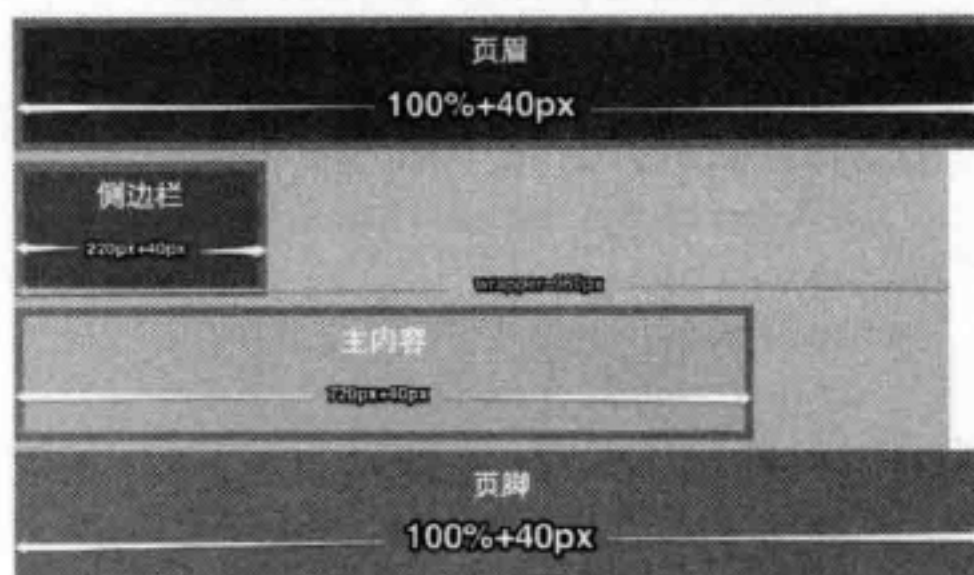


图 7-7 padding 和 border 撑破布局

```

background: #c8ca30;
margin-bottom: 10px;
border: 10px solid red;
padding: 10px;
-moz-box-sizing: border-box;
-webkit-box-sizing: border-box;
-o-box-sizing: border-box;
-ms-box-sizing: border-box;
box-sizing: border-box;
}
#footer {
background: #cc4ad5;
height: 100px;
text-align: center;
clear: both;
border: 10px solid red;
padding: 10px;
width: 100%;
-moz-box-sizing: border-box;
-webkit-box-sizing: border-box;
-o-box-sizing: border-box;
-ms-box-sizing: border-box;
box-sizing: border-box;
}

```

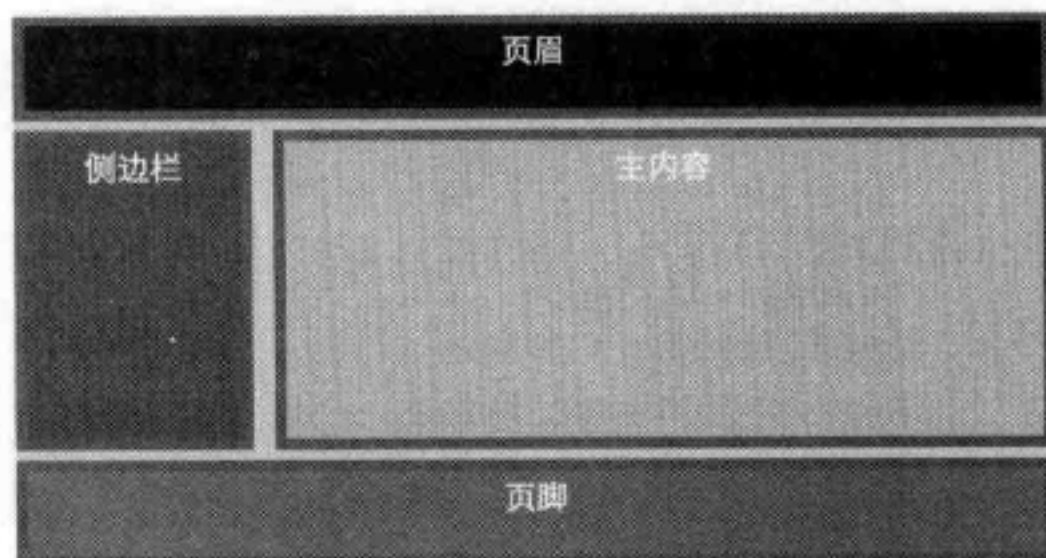


图 7-8 box-sizing 修复布局效果

修复后的布局如图 7-8 所示。

通过对 #header、#footer、.sidebar 和 .content 等元素显式设置 box-sizing 属性为 border-box 值，从而修改了盒模型的解析模式。经过这样设置后，整个布局神奇般的好了。以前需要改变 box 的宽度或者在 box 里嵌套一个 div，在 div 中增加 padding 和 border 来达到这样的效果。现在开始，只要通过 box-sizing 重置盒模型的解析模式为 border-box 回到 IE 的传统模式下，就可以实现了。遗憾的是，在 IE 6 和 IE 7 不支持 box-sizing 属性，但是如果在 IE 6 和 IE 7 对元素宽度做一定的修改，也能达到效果。

```

.sidebar {
width: 180px;
}
.content {
width: 680px;
}
#header, #footer {
width: auto;
}

```

这种方式针对准确像素尺寸来说还好计算，如果是以百分比为单位的布局，要重新为 IE 7 及以下版本计算盒模型尺寸就会存在一定的难度。因为百分比和像素两者之间是无法直接进行换算。

7.3 CSS3 内容溢出属性

在 CSS 中的每一个元素都视为一个盒子，这个盒子就是一个容器。容器就有大小之分，如果在样式中指定盒子的宽度与高度，可能某些内容在盒子中就会容纳不下，此时就可以使用 `overflow` 属性来指定如何显示盒中容纳不下的内容。`overflow` 属性是 CSS2.1 规范中的特性，而在 CSS3 中增加了 `overflow-x` 和 `overflow-y` 属性。

7.3.1 `overflow-x` 和 `overflow-y` 属性的语法及参数

`overflow-x` 主要是用来定义对水平方向内容溢出的剪切，而 `overflow-y` 主要用来定义对垂直方向内容溢出的剪切。`overflow-x` 和 `overflow-y` 属性的基本语法如下。

```
overflow-x: visible | hidden | scroll | auto | no-display | no-content
overflow-y: visible | hidden | scroll | auto | no-display | no-content
```






和 `overflow` 属性参数一样，`overflow-x` 和 `overflow-y` 属性值取不同的值所起的作用不一样。

- ❑ `visible`：默认值。表示不剪切容器中的任何内容、不添加滚动条，元素将被剪切为包含对象的窗口大小，而且 `clip` 属性设置将失效。
- ❑ `auto`：在需要时剪切内容并添加滚动条。也就是说当内容超过容器的宽度或者高度时，溢出的内容将会隐藏在容器中，并且会添加滚动条，用户可以拖动滚动条查看隐藏在容器中的内容。
- ❑ `hidden`：内容溢出容器时，所有内容都将隐藏，而且不显示滚动条。
- ❑ `scroll`：不管内容有没有溢出容器，`overflow-x` 都会显示横向的滚动条，而 `overflow-y` 会显示纵向的滚动条。
- ❑ `no-display`：当内容溢出容器时不显示元素，此时类似于元素添加了 `display:none` 声明一样。
- ❑ `no-content`：当内容溢出容器时不显示内容，此时类似于添加了 `visibility:hidden` 声明一样。

7.3.2 浏览器兼容性

`overflow-x` 和 `overflow-y` 原本是 IE 浏览器中独自扩展的属性，后来被 CSS3 采用，并标准化。目前为止，所有浏览器都能够正确解析这两个属性，但是部分浏览器在解析时，会存在一些细节上的差异，稍后会做进一步测试分析。浏览器对 `overflow-x` 和 `overflow-y` 属性的支持情况如表 7-2 所示。

表 7-2 `overflow-x` 和 `overflow-y` 浏览器兼容性

属性名					
<code>overflow-x</code>	6+ ✓	1.5+ ✓	1.0+ ✓	9.0+ ✓	3.0+ ✓
<code>overflow-y</code>	6+ ✓	1.5+ ✓	1.0+ ✓	9.0+ ✓	3.0+ ✓

7.4 CSS3 自由缩放属性

为了增强用户体验，CSS3 增加了很多新的属性，其中 `resize` 就是一个重要的属性，也是一个非常实用的属性，它允许用户通过拖动的方式来修改元素的尺寸来改变元素的大小。到目前为止，可以使用 `overflow` 属性的任何容器元素。在此之前，Web 设计师为了要实现这样的 UI 效果，需要使用大量的脚本代码才能实现，这样费时费力，效率极低。

7.4.1 `resize` 属性的语法及参数

`resize` 属性主要是用来改变元素尺寸大小的，其主要目的是增强用户体验。使用方法极其简单。

```
resize: none | both | horizontal | vertical | inherit
```






在 CSS3 中 `resize` 属性指定的值分为以下几种。

- ❑ `none`: 用户不能拖动元素修改尺寸大小。
- ❑ `both`: 用户可以拖动元素，同时修改元素的宽度和高度。
- ❑ `horizontal`: 用户可以拖动元素，仅可以修改元素的宽度，但不能修改元素的高度。
- ❑ `vertical`: 用户可以拖动元素，仅可以修改元素的高度，但不能修改元素的宽度。
- ❑ `inherit`: 继承父元素的 `resize` 属性值。

7.4.2 浏览器兼容性

到目前为止，除了 IE 浏览器之外，`resize` 属性得到了 Firefox、Safari、Chrome 以及 Opera 等主流浏览器的支持，如表 7-3 所示。

表 7-3 `resize` 浏览器兼容性

属性名					
resize	×	19.0+ ✓	25.0+ ✓	12.0+ ✓	5.0+ ✓

7.4.3 实战体验：修改文本域随意调整大小的功能

表单中的文本域 `textarea` 元素默认情况就具有 `resize` 功能，可以通过按住 `textarea` 右下角进行拖放，放大或者缩小 `textarea` 元素大小。

`textarea` 默认具有 `resize` 属性，其默认属性值为 “`both`”，按住 `textarea` 元素右下角部分拖动，可以修改 `textarea` 元素大小（改变 `textarea` 的高度和宽度）。但在实际项目中，有时候仅需要拖动 `textarea` 来修改其宽度、高度，有时候需要固定 `textarea` 大小，不允许拖动 `textarea` 来修改其宽度和高度。实现这样的效果，完全可以通过改变 `resize` 属性值来达到设计所需的效果，如图 7-9 所示。



拖动此处放大、缩小文本域

图 7-9 textarea 元素默认的 resize 效果

下面在浏览器中实现 textarea 固定大小，禁止用户拖动 textarea 改变大小。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title> 修改文本域随意调整大小的功能 </title>
  <style type="text/css" media="screen">
    textarea {
      resize: none;
    }
  </style>
</head>
<body>
  <form action="#" method="post">
    <textarea name="test" rows="10" cols="30"
      placeholder=" 修改文本域随意调整大小的功能"
    ></textarea>
  </form>
</body>
</html>
```

上面的代码说明，只需要把 textarea 元素的 resize 属性值设置为 none，就可以轻松地实现所需要的效果，如图 7-10 所示。



无法拖动，修改大小

图 7-10 textarea 元素设置 resize 属性值为 none 的效果

7.5 CSS3 外轮廓属性

外轮廓 outline 在页面中呈现的效果和边框 border 呈现的效果极其相似，但和元素边框 border 完全不同，外轮廓线不占用网页布局空间，不一定是矩形，外轮廓是属于一种动态样式，只有元素获取到焦点或者被激活时呈现。

7.5.1 outline 属性的语法及参数

outline 属性早在 CSS2 中就出现了，主要是用来在元素周围绘制一条轮廓线，可以起

到突出元素的作用，但是并未得到各主流浏览器的广泛支持。在 CSS3 中对 outline 作了一定的扩展，在以前的基础上增加新特性。outline 属性的基本语法如下。

```
outline: [outline-color] || [outline-style] || [outline-width] || [outline-offset]
|| inherit
```






从语法中可以看出 outline 和 border 边框属性的使用方法极其类似。outline-color 相当于 border-color，outline-style 相当于 border-style，而 outline-width 相当于 border-width，只不过 CSS3 给 outline 属性增加了一个 outline-offset 属性，其取值说明如下。

- ❑ outline-color：定义轮廓线的颜色，属性值为 CSS 中定义的颜色值。在实际应用中，省略时此参数的默认值为黑色。
- ❑ outline-style：定义轮廓线的样式，属性为 CSS 中定义线的样式。在实际应用中，省略时此参数的默认值为 none，省略后不对该轮廓线进行任何绘制。
- ❑ outline-width：定义轮廓线的宽度，属性值可以为一个宽度值。在实际应用中，省略时此参数的默认值为 medium，表示绘制中等宽度的轮廓线。
- ❑ outline-offset：定义轮廓边框的偏移位置的数值，此值可以取负数值。当此参数的值为正数值，表示轮廓边框向外偏离多少个像素；当此参数的值为负数值，表示轮廓边框向内偏移多少个像素。
- ❑ inherit：元素继承父元素的 outline 效果。

7.5.2 浏览器兼容性

只有 IE 7 及以下版本浏览器不支持，其他浏览器都支持 outline（见表 7-4）。

表 7-4 outline 浏览器兼容性

属性名					
outline	8+ ✓	1.5+ ✓	1.0+ ✓	9.0+ ✓	3.0+ ✓

7.5.3 outline 和 border 的对比

outline 属性创建的外轮廓线外表上和 border 极其相似，但实际上有明显的不同。

- ❑ border 属于盒模型的一部分，直接影响元素盒子的大小，而 outline 创建的外轮廓线是画在一个框的“上面”，不会影响该框或任何其他框大小，因此 outline 创建的轮廓线不会影响文档流，也不会破坏网页布局。
- ❑ outline 创建的轮廓线表面上和 border 一样，可以创建轮廓线的颜色、线型样式、线型粗细大小，但和 border 有一点完全不一样。outline 创建的外轮廓线在元素各边都一样，这和 border 不一样，不能像 border 边框一样，设置 outline-top 或 outline-left 之类。
- ❑ border 创建的元素边框可以单边设置，而 outline 创建的外轮廓线始终是闭合的。
- ❑ outline 创建的外轮廓线可能是非矩形的，如果元素是多行，外轮廓线就至少是能够包含该元素所有框的外轮廓。可 border 不一样，他将使用一个边框包括整个元素。

- border 仅可以设置元素的边框，只能向外扩展，而 outline 创建的外轮廓线，可以通过 outline-offset 的值，向元素外部（outline-offset 值为正值）或向元素内部（outline-offset 值为负值）创建封闭的轮廓。

7.5.4 实战体验：模仿边框效果

说起边框效果的制作，大家首先想起的就是 border 属性，但 border 属性制作边框效果，会影响元素的盒模型大小，也就会影响文档流，甚至会直接影响页面的布局。

为了给元素设置边框效果，又保证添加边框后不影响页面的布局，使用 outline 属性模仿 border 制作边框效果是最完美不过了。效果如图 7-11 所示。

使用 outline 属性给 div 元素设置一个边框效果代码如下。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>outline 模仿边框效果 </title>
  <style type="text/css">
    div {
      width: 200px;
      margin: 20px auto;
      outline: 10px solid #ccc;
      padding: 20px;
    }
  </style>
</head>
<body>
  <div>outline 模仿边框效果 </div>
</body>
</html>
```

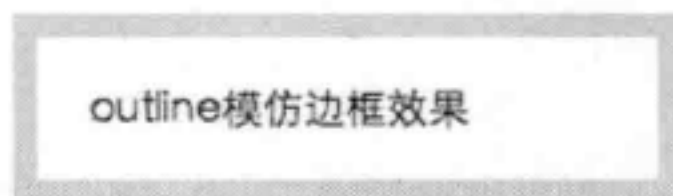


图 7-11 outline 模拟边框效果

上例中，直接使用 outline 模拟了一个 10px 的边框，从外表上看和 border 制作的效果可以说是完全一样，唯一不同的是，outline 制作的边框只能同时四边出现，不能单边出现，而且 outline 制作的模拟边框不会影响盒模型大小，而 border 制作的边框直接影响元素盒模型大小。

7.6 本章小结

本章先简单回顾了 CSS 中的盒模型相关知识，然后介绍了 CSS3 为盒模型新增的属性以及直接影响用户界面的新功能。box-sizing 属性可以显式的重置 W3C 标准规范中的盒模型解析模式，从而可以利用此属性来修补一些因为盒模型其他属性而破坏的布局；使用 overflow-x 或者 overflow-y 单方面的控制元素内容溢出的处理方案；使用 resize 属性来控制拖放元素而修改元素尺寸大小；使用 outline 来控制元素得到焦点时的样式效果，并且可以使用 outline 模仿边框 border 的效果。

CSS3 伸缩布局盒模型

早期的布局主要依赖于表格，从 CSS2.1 中有关于布局的机制有所改变，但还是并不多。开发者通常不愿意使用绝对定位，因为太不灵活；浮动定位常用于页面的布局，但对于全页多列布局来说，它总是存在一些小毛病，功能上也有很多限制。CSS3 中引入了许多的布局机制，使构建一个多列布局变得更加轻松，同时，CSS2.1 规则是比较难实现的一些复杂布局表现，如今也变得更加容易。在本章将了解未来的布局技术发展趋势，同时学习到如何使用伸缩布局盒模型的实战技巧。

8.1 Flexbox 模型基础知识

CSS3 引入一种新的布局模式——Flexbox 布局，即伸缩布局盒（Flexible Box）模型，用来提供一个更加有效的方式制定、调整和分布一个容器里的项目布局，即使它们的大小是未知或者动态的，这里简称 Flex。

8.1.1 CSS 中的布局模式

谈到布局，CSS2.1 中定义了四种布局模式，由一个盒与其兄弟、祖先盒的关系决定其尺寸与位置的算法。

- 块布局：呈现文档的布局模式。
- 行内布局：呈现文本的布局模式。
- 表格布局：用格子呈现 2D 数据的布局模式。
- 定位布局：能够直接地定位元素的布局模式，定位元素基本与其他元素没有任何关系。

CSS3 引入的布局模式 Flexbox 布局, 主要思想是让容器有能力让其子项目能够改变其宽度、高度 (甚至顺序), 以最佳方式填充可用空间 (主要是为了适应所有类型的显示设备和屏幕大小)。Flex 容器会使子项目 (伸缩项目) 扩展来填满可用空间, 或缩小它们以防止溢出容器。

最重要的是, Flexbox 布局方向不可预知, 不像常规的布局 (块就是从上到下, 内联就从左到右), 而那些常规的适合页面布局, 但对于支持大型或者复杂的应用程序 (特别是涉及取向改变、缩放和收缩等) 就缺乏灵活性。

8.1.2 Flexbox 模型的功能

Flexbox 布局对于设计比较复杂的页面非常有用。可以轻松实现屏幕和浏览器窗口大小发生变化时保持元素的相对位置和大小不变。同时减少了依赖于浮动布局实现元素位置的定义以及重置元素的大小。

Flexbox 布局在定义伸缩项目大小时伸缩容器会预留一些可用空间, 可以调节伸缩项目的相对大小和位置。例如, 可以确保伸缩容器中的多余空间平均分配多个伸缩项目。当然, 如果伸缩容器没有足够大的空间放置伸缩项目时, 浏览器会根据一定的比例减少伸缩项目的大小, 使其不溢出伸缩容器。

综合而言, Flexbox 布局功能主要具有以下几点。

- ❑ 屏幕和浏览器窗口大小发生改变也可以灵活调整布局。
- ❑ 指定伸缩项目沿着主轴或侧轴按比例分配额外空间 (伸缩容器额外空间), 从而调整伸缩项目的大小。
- ❑ 指定伸缩项目沿着主轴或侧轴将伸缩容器额外空间, 分配到伸缩项目之前、之后或之间。
- ❑ 指定如何将垂直于元素布局轴的额外空间分布到该元素的周围。
- ❑ 控制元素在页面上的布局方向。
- ❑ 按照不同于文档对象模型 (DOM) 所指定排序方式对屏幕上的元素重新排序。也就是说可以在浏览器渲染中不按照文档流先后顺序重排伸缩项目顺序。

8.1.3 Flexbox 模型中的术语

和 CSS3 其他属性不一样, Flexbox 并不是一个属性, 而是一个模块, 包括多个 CSS3 属性, 涉及很多东西, 包括整个组属性。虽然现在对 Flexbox 有一定的了解, 如果想更好地使用 Flexbox, 新的术语和概念可能是一个障碍, 所以首先了解基本概念。

1. 主轴和侧轴

在 Flexbox 模型中与布局计算偏向使用书写模式方向的块布局与行内布局不同, 伸缩布局偏向使用伸缩流的方向。为了让描述伸缩布局变得更容易, 本章定义一系列相对于伸缩

流的术语。如图 8-1 所示。

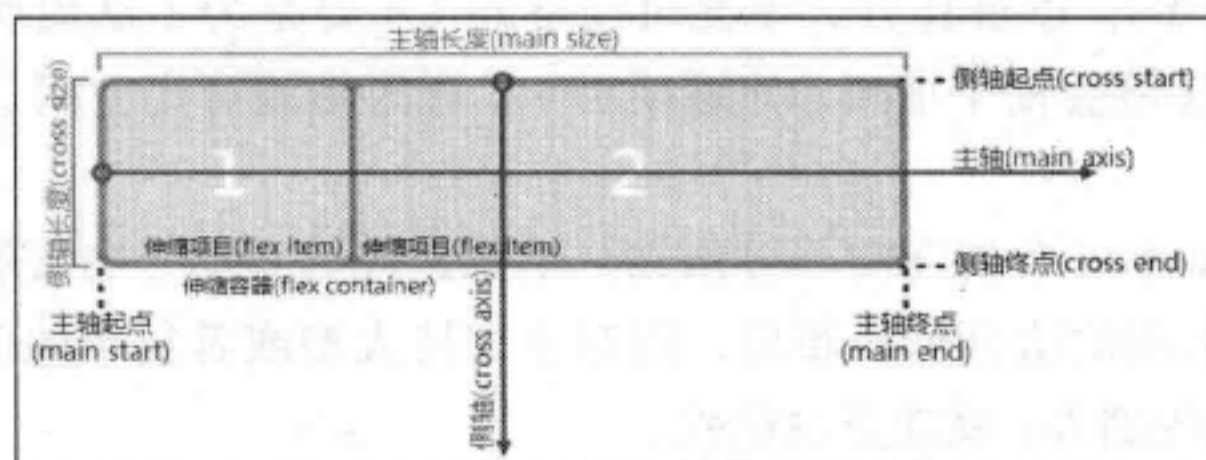


图 8-1 一个 row 伸缩容器中各种方向与大小术语

- ❑ **主轴、主轴方向：**用户代理沿着一个伸缩容的主轴配置伸缩项目，主轴是主轴方向的延伸。伸缩容器的主轴，伸缩项目主要沿着这条轴进行布局。小心，它不一定是水平的，这主要取决于 justify-content 属性。如果其取值为 column，主轴的方向为纵向的。
- ❑ **主轴起点、主轴终点：**伸缩项目的配置从容器的主轴起点边开始，往主轴终点边结束。也就是说，伸缩项目放置在伸缩容器内从主轴起点（main-start）向主轴终点（main-end）方向。
- ❑ **主轴长度、主轴长度属性：**伸缩项目的在主轴方向的宽度或高度就是项目的主轴长度，伸缩项目的主轴长度属性是 width 或 height 属性，由哪一个对着主轴方向决定。
- ❑ **侧轴、侧轴方向：**与主轴垂直的轴称做侧轴，侧轴是侧轴方向的延伸。主要取决于主轴方向。
- ❑ **侧轴起点、侧轴终点：**填满项目的伸缩行的配置从容器的侧轴起点边开始，往侧轴终点边结束。
- ❑ **侧轴长度、侧轴长度属性：**伸缩项目的在侧轴方向的宽度或高度就是项目的侧轴长度，伸缩项目的侧轴长度属性是 width 或 height 属性，由哪一个对着侧轴方向决定。

2. 伸缩容器和伸缩项目

通过 display 属性显式地给一个元素设置为 flex 或者 inline-flex，这个容器就是一个伸缩容器。伸缩容器会为其内容创立新的伸缩格式化上下文，其中设置为 flex 的容器被渲染为一个块级元素，而设置为 inline-flex 的容器则渲染为一个行内元素。若元素 display 的指定值是 inline-flex 且元素是一个浮动或绝对定位元素，则 display 的计算值是 flex。

一个伸缩容器的内容具有零个以上的伸缩项目——伸缩容器的每个子元素（除了需要盒修复的元素之外）都会成为一个伸缩项目，且用户代理会将任何直接在伸缩容器里的连续文字块包起来成为匿名伸缩项目。

3. 伸缩容器的属性

Flexbox 伸缩布局中伸缩容器和伸缩项目是伸缩布局模块中的重要部分，其中每一部分都具有各自的属性。伸缩容器的属性包括以下几个。

1) **伸缩流方向**。是指伸缩容器的主轴方向,其决定了伸缩项目放置在伸缩容器的方向。伸缩流方向主要通过 `flex-direction` 属性来设置,其默认值为 `row`。伸缩流方向和书写模式有关系,换句话说,伸缩项目根据书写模式的方向布局。



注意 在 2009 版本伸缩流方向主要通过 `box-orient` 属性来设置,初始值为 `inline-axis`

2) **伸缩行换行**。伸缩项目在伸缩容器中有时候也会溢出伸缩容器。与传统的盒模型一样,CSS 允许使用 `overflow` 属性来处理溢出内容的显示方式。在伸缩容器中有一个伸缩换行属性,主要用来设置伸缩容器的伸缩项目是单行显示还是多行显示,以及决定侧轴的方向。使用此属性来启用溢出的伸缩容器的伸缩项目自动换行到下一行以及控制流动方向。在伸缩容器属性中,主要通过 `flex-wrap` 属性来设置伸缩项目是否换行,默认值为 `nowrap`。



注意 在 2009 版本语法中,伸缩行换行主要通过 `box-lines` 属性来设置,默认值为 `single`。

3) **伸缩方向与换行**。是伸缩流方向与伸缩换行的结合物,换句话说,伸缩方向与换行属性 `flex-flow` 同时设定了伸缩流方向 `flex-direction` 和伸缩换行 `flex-wrap` 两个属性,简而言之是这两个属性的缩写,这两个属性决定了伸缩容器的主轴与侧轴。

4) **主轴对齐**。用来设置伸缩容器当前行伸缩项目在主轴方向的对齐方式,指定如何在伸缩项目之间分布伸缩容器额外空间。当一行上的所伸缩项目不能伸缩或可伸缩但已达到最大长度时,这一属性才会对伸缩容器额外空间进行分配。当伸缩项目溢出某一行时,这一属性也会在项目的对齐上施加一些控制。

5) **侧轴对齐**。伸缩项目可以在伸缩容器当前行的侧轴上进行对齐,这类似于主轴对齐方式,只是另一个方向。也就是说侧轴对齐主要用来指定伸缩项目在伸缩容器中如何放置和对齐的方式。换句话说,用来定义伸缩项目在伸缩容器的当前行的侧轴上对齐方式。

6) **堆栈伸缩行**。用来定义伸缩容器中伸缩项目行在侧轴的对齐方式,类似于侧轴对齐,只不过这是用来控制伸缩项目行在布局轴的堆放方式。

4. 伸缩项目的属性

一个伸缩项目是一个伸缩容器的子元素。伸缩容器中的文本也被视为一个伸缩项目。伸缩项目中内容与普通流一样。比如说,当一个伸缩项目被设置为浮动,依然可以在这个伸缩项目中放置一个浮动元素。

伸缩项目都有一个主轴长度和一个侧轴长度。主轴长度是伸缩项目在主轴上的尺寸,侧轴长度是伸缩项目在侧轴上的尺寸。或者说,一个伸缩项目的宽度或高取决于伸缩容器的轴,可能就是它的主轴长度或侧轴长度。下面的几个属性可以调整伸缩项目的行为。

1) **显示顺序**。伸缩容器中的伸缩项目默认显示顺序是遵循文档在源码中的出现的先后顺序(HTML 文档的 DOM 结构中的先后顺序)。可以通过伸缩项目的显示顺序修改伸缩项目在页面中显示顺序,也可以通过这个属性对伸缩项目进行排序组合。

2) **侧轴对齐**。包括两种,一种是 `align-items` 属性,可以用来设置伸缩容器中包括匿名伸缩项目的所有项目的对齐方式;另一种是 `align-self` 属性,主要用来在单独的伸缩项目上覆盖默认的对齐方式。对于匿名伸缩项目, `align-self` 的值永远与其关联的伸缩容器的 `align-items` 的值相同。

3) **伸缩性**。定义伸缩项目可改变它们的宽度或高度填补可用的空间。可以将伸缩容器的额外空间分发给其伸缩项目(与伸缩项目的正弹性成正比)或将它们缩小以防止伸缩项目溢出(与伸缩项目的负弹性成正比)。

5. 伸缩行

伸缩项目沿着伸缩容器内的一个伸缩行定位。伸缩容器可以是单行的,也可以是多行的。其主要由 `flex-wrap` 属性决定。单行的伸缩容器会将其所有子元素在单独的一行上进行布局,这种情况之下有可能导致内容溢出伸缩容器;多行的伸缩窗口会将其伸缩项目配置在多个伸缩行上,这类似于文本的排列。当文本过宽导致一行无法容纳时,内容会断开并移至新的一行。当用户代理创建新的伸缩行时,这些伸缩行会根据 `flex-wrap` 属性沿着侧轴进行堆叠。除非伸缩容器本身是空的,每一个伸缩行至少包含一个伸缩项目。

8.1.4 Flexbox 模型规范状态

Flexbox 布局的语法规则经过几年发生了很大的变化,也给 Flexbox 的使用带来一定的局限性,因为语法规则版本众多,浏览器支持不一致,致使 Flexbox 布局使用不多。

Flexbox 语法规则主要经历了以下阶段。

- 1) 2009 年 07 月工作草案 (`display:box`);
- 2) 2011 年 03 月工作草案 (`display:flexbox`);
- 3) 2011 年 11 月工作草案 (`display:flexbox`);
- 4) 2012 年 03 月工作草案 (`display:flexbox`);
- 5) 2012 年 06 月工作草案 (`display:flex`);
- 6) 2012 年 09 月候选推荐 (`display:flex`)。

把 Flexbox 布局语法规则主要分成三种。

- ☐ 旧版本 (Old), 2009 年版本, 使用 `display:box` 或者 `display:inline-box`。
- ☐ 混合版本 (Hybrid), 2011 年版本, 使用 `display:flexbox` 或者 `display:inline-flexbox`。
- ☐ 最新版本 (Modern), 使用 `display:flex` 或者 `display:inline-flex`。






8.1.5 Flexbox 模型浏览器兼容性

Flexbox 规范版本众多,浏览器对此语法支持度也各有不同,下面针对不同语法规则版本的浏览器兼容性做一个对比。

1. 旧版本 Flexbox 模型浏览器支持情况

Flexbox 布局的旧版本语法规则是最早的伸缩布局，各大主流浏览器对其支持性略有不同，可惜的是，各浏览器对其 Flexbox 布局的各属性支持并不完全，见表 8-1。

表 8-1 Flexbox 旧版本的浏览器兼容性






属性					
Flexbox	×	2 ~ 21 ✓	4 ~ 20 ✓	×	3.1 ~ 6 ✓

对于 Flexbox 旧版本，各大浏览器以及手持设备支持都不完全支持 Flexbox 模块的所有属性，而且在使用时都必须在前面添加各浏览器的前缀。


2. 新版本 Flexbox 模型浏览器支持情况

到目前为止除了 Safari 浏览器不支持 Flexbox 新版本之外，其他浏览器最新版本都将支持这个版本语法，如表 8-2 所示。

表 8-2 Flexbox 新版本的浏览器兼容表

属性					
Flexbox	11 + ✓	22 + ✓	21 + ✓	12.1 + ✓	×

不过在 Webkit 内核浏览器还需要使用其前缀“-webkit-”。在手持设备中，到目前为止仅“Opera Mobile 12.1+”支持 Flexbox 最新版本的语法。

 注意 混合版本 Flexbox 浏览器兼容性相对来说要简单，因为一直以来仅 IE10 支持这个版本。

8.1.6 Flexbox 模型语法变更

表 8-3 ~ 表 8-12 说明了各版本之间的变更情况。

表 8-3 Flexbox 模型语法规则版本

规范版本	IE	Opera	Firefox	Chrome	Safari
标准版本	11 ?	12.10+*		21+(-webkit-)	
混合版本	10(-ms-)				
最老版本			3+(-moz-)	<21(-webkit-)	3+(-webkit-)

Flexbox 模型各版本规范在各浏览器下的兼容性，详细可以参阅 Flexbox 模型浏览器支持情况。

表 8-4 开启 Flexbox：让一个元素变成伸缩容器

规范版本	属性名称	块伸缩容器	内联伸缩容器
标准版本	display	flex	inline-flex
混合版本	display	flexbox	inline-flexbox
最老版本	display	box	inline-box

Flexbox 模型各版本规范开启 Flexbox，换句话说，让一个元素变成伸缩容器，都是通过 display 属性来设置。当一个元素变成了伸缩容器，其子元素会自动变成伸缩项目。

表 8-5 主轴对齐方式：指定伸缩项目沿主轴对齐方式

规范版本	属性名称	start	center	end	justify	distribute
标准版本	justify-pack	flex-start	center	flex-pack	flex-end	space-around
混合版本	flex-pack	start	center	end	justify	distribute
最老版本	box-pack	start	center	end	justify	N/A

主轴对齐方式主要用来指定伸缩项目沿主轴对齐方式，主要包括左边对齐、中间对齐、右对齐、两端对齐和扩散对齐，这几种对齐方式可以将其理解为 Word 文本编辑器中的对齐方式。

表 8-6 侧轴对齐方式：指定伸缩项目沿侧轴对齐方式

规范版本	属性名称	start	center	end	baseline	stretch
标准版本	align-items	flex-start	center	flex-end	baseline	stretch
混合版本	flex-align	start	center	end	baseline	stretch
最老版本	box-align	start	center	end	baseline	stretch

侧轴对齐方式主要用来指定伸缩项目沿侧轴的对齐方式，主要包括顶边对齐、中间对齐、底部对齐、基线对齐和伸缩项目拉伸填充整个伸缩容器。

表 8-7 伸缩项目行对齐方式：指定伸缩项目行在侧轴的对齐方式

规范版本	属性名称	start	center	end	justify	distribute	stretch
标准版本	align-content	flex-start	center	flex-end	space-between	space-around	stretch
混合版本	flex-line-pack	start	center	end	justify	distribute	stretch
最老版本	N/A						

这种情况只有伸缩容器设置 flex-wrap 为 wrap 时，并且没有足够的空间把伸缩项目放在同一行中。也就是说这个属性只有伸缩项目有多行时才生效。

表 8-8 单个伸缩项目对齐方式：指定单个伸缩项目在侧轴方式

规范版本	属性名称	auto	start	center	end	baseline	stretch
标准版本	align-self	auto	flex-start	center	flex-end	baseline	stretch
混合版本	flex-item-align	auto	start	center	end	baseline	stretch
最老版本	N/A						

这个属性类似侧轴对齐方式属性，主要用来设置伸缩项目在侧轴的对齐方式，用来覆盖伸缩项目在伸缩容器中侧轴的对齐方式。

表 8-9 显示顺序：指定伸缩项目的顺序

规范版本	属性名称	属性值
标准版本	order	<number>
混合版本	flex-order	<number>
最老版本	box-ordinal-group	<integer>

这个属性主要用来重置伸缩项目的显示顺序。

表 8-10 伸缩性：指定伸缩项目如何伸缩尺寸

规范版本	属性名称	属性值
标准版本	flex	none [<flex-grow> <flex-shrink> ? <flex-basis>]
混合版本	flex	none [<pos-flex> <neg-flex> ? <preferred-size>]
最老版本	box-flex	<number>

伸缩性属性主要用来控制伸缩项目在伸缩容器中扩展或收缩伸缩项目的宽度，用来填充伸缩容器的额外空间。

表 8-11 伸缩流：指定伸缩容器主轴的伸缩流方向

规范版本	属性名称	水平方向	反向水平	垂直方向	反向垂直
标准版本	flex-direction	row	row-reverse	column	column-reverse
混合版本	flex-direction	row	row-reverse	column	column-reverse
最老版本	box-orient	horizontal	horizontal	vertical	vertical
	box-direction	normal	reverse	normal	reverse

这个属性主要用来控制伸缩项目在没有足够空间的伸缩容器中是否换行。

表 8-12 换行：指定伸缩项目是否沿着侧轴排列

规范版本	属性名称	不换行	换行	反转换行
标准版本	flex-wrap	nowrap	wrap	wrap-reverse
混合版本	flex-wrap	nowrap	wrap	wrap-reverse
最老版本	box-lines	single	multiple	N/A

扩展阅读 如何辨别旧 Flexbox 和新 Flexbox

如果使用搜索引擎搜索“Flexbox”，会发现很多过时的信息。这里将告诉你如何迅速辨别信息。

- ❑ 看到“display:box”或者“box-{}”属性，说明是 2009 年版本的 Flexbox。
- ❑ 看到“display:flexbox”或者“flex()”函数，说明是 2011 年版本，也称为 Flexbox 混合版本。
- ❑ 看到“display:flex”或者“flex-{}”属性，说明是当前规范，也就是 W3C 标准规范版本的 Flexbox。

8.2 旧版本 Flexbox 模型的基本使用

Flexbox 的语法规则多样性造成在不同的语法版本下使用方式也不一样，再加上浏览器对各语法的支持度不一样，致使需要了解各版本下的使用。接下来分别按照 Flexbox 的语法分类，依次介绍各种版本的伸缩布局的使用。

为了让 Safari 浏览器下能正常使用 Flexbox 模型，需要掌握旧版本 Flexbox 模型的基本使用。

8.2.1 伸缩容器设置 display

要改变元素的模式，需要使用 display 属性，如果在让一个元素变成伸缩容器，也离不开 display 属性。

1. display 属性的语法及参数

伸缩容器设置语法很简单，只需要将 display 显式设置为 box 或者 inline-box 属性值。

```
display: box | inline-box
```

属性值主要有两种。

- ❑ box：设置为块伸缩容器。
- ❑ inline-box：设置为内联级伸缩容器。

2. display 属性的基本使用

伸缩容器主要用来将元素设置为伸缩容器。伸缩容器为其内容创建新的伸缩格式化上下文（Flex formatting context）。除了伸缩排版用来代替块布局以外，创建一个伸缩格式化上下文与创建一个块格式化上下文格式是一样的。浮动无法影响伸缩容器，而且伸缩容器的 margin 与其内容的 margin 不会重叠。如果给浮动元素或者绝对定元素设置了内联伸缩容器，元素将会以块伸缩容器展示。

如果想把某个元素变成伸缩容器，只需要给这个元素显式设置 display 值，并且将其值设置为 box 或者 inline-box。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title> 将容器设置为伸缩容器 </title>
  <style type="text/css" media="screen">
    *{
      margin: 0;
      padding: 0;
    }
    body>div {
      border: 1px solid #ccc;
      margin: 20px;
      padding: 10px;
    }
    div > div {
      border: 1px solid #f36;
      width: 100px;
      height: 100px;
```

```

        text-align: center;
        line-height: 100px;
    }
    .box {
        display: -moz-box;
        display: -webkit-box;
        display: box;
    }
    .inline-box {
        display: -moz-inline-box;
        display: -webkit-inline-box;
        display: inline-box;
    }
</style>
</head>
<body>
    <div class="box">
        <div>A</div>
        <div>B</div>
        <div>C</div>
        <div>D</div>
    </div>
    <div class="inline-box">
        <div>A</div>
        <div>B</div>
        <div>C</div>
        <div>D</div>
    </div>
</body>
</html>

```

其效果如图 8-2 所示。

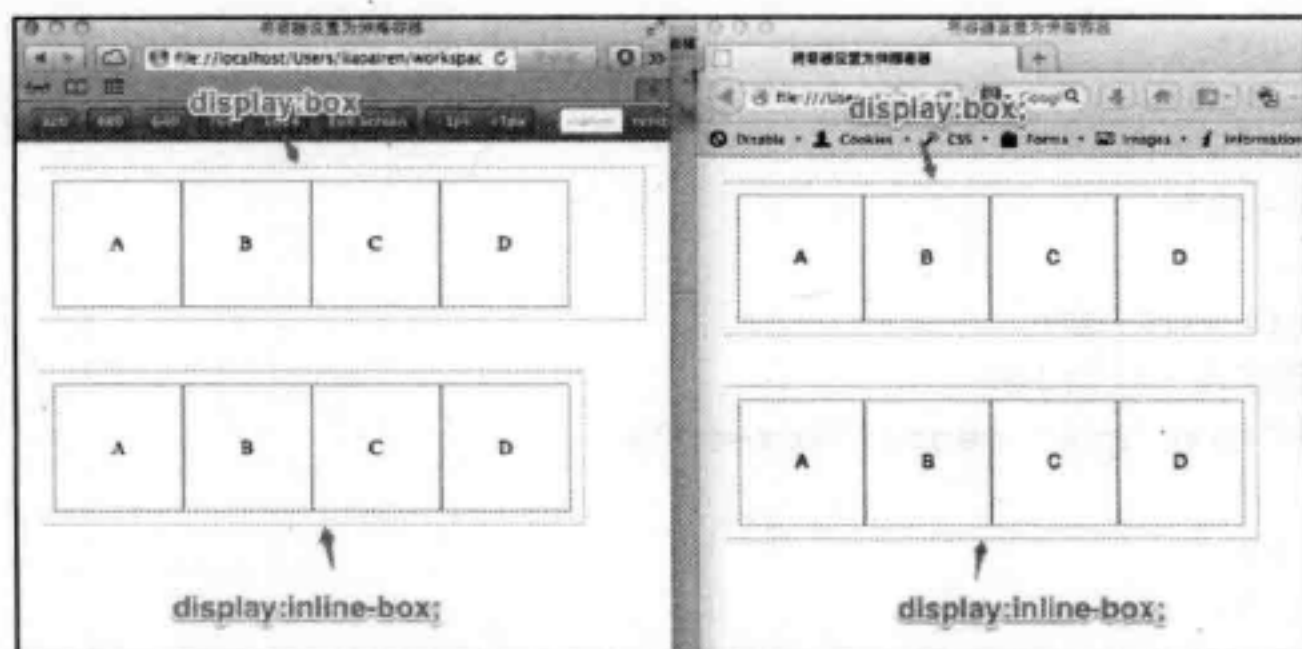


图 8-2 Safari 和 Firefox 下伸缩器的设置效果

图 8-2 的效果明显可知，如果给元素设置块伸缩容器，其效果类似于块元素的渲染风格，不过在 Firefox 浏览器下和 inline-block 风格一致；如果给元素设置内联级伸缩容器，其效果类似于 inline-block 风格。

8.2.2 伸缩流方向 box-orient

伸缩流方向 box-orient 属性主要用来创建主轴，从而定义了伸缩项目在伸缩容器中的流动布方向。换句话说主要用来指定伸缩项目如何放置在伸缩容器的主轴上。

1. box-orient 属性的语法及参数

给伸缩项目确定其在伸缩容器中的流动方向，通过 box-orient 属性来设置，其语法如下所示。

```
box-orient: horizontal | vertical | inline-axis | block-axis
```

伸缩流方向 box-orient 主要适用于伸缩容器。伸缩流方向主要是用来确定伸缩项目在伸缩容器中的流动布局方向，该属性主要有四个属性值，其取值说明如下。

- ❑ horizontal: 伸缩项目在伸缩容器中从左到右在一条不平线上排列显示。
- ❑ vertical: 伸缩项目在伸缩容器中从上到下在一条垂直线上排列显示。
- ❑ inline-axis: 伸缩项目沿着内联轴排列显示。
- ❑ block-axis: 伸缩项目沿着块轴排列显示。

伸缩流方向和文本书写模式也有关系，如果书写模式是 ltr，表示排版本方向从左向右，如果书写模式是 rtl 表示排版方向从右向左排列。伸缩流的取值为 horizontal 和 inline-axis 时，确认伸缩项目的伸缩流方向和书写模式的方向关连性非常的强。可以说书写模式直接影响了它们的排列方向。

2. box-orient 属性的基本使用

在传统的盒模型中，HTML 的文档流（DOM）总是按先后顺序从上到下垂直排列。使用伸缩布局的伸缩流方向，可以重新设置其文档流的方向。前提条件是要开启 Flexbox 模型。基于前面的实例上，分别在 Flexbox 模型上设置伸缩流方向为 horizontal、vertical、inline-axis 和 block-axis。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title> 伸缩流方向 </title>
  <style type="text/css" media="screen">
    *{
      margin: 0;
      padding: 0;
    }
    body>div {
      border: 1px solid #ccc;
      margin: 20px;
      padding: 10px;
    }
    div > div {
```

```
border: 1px solid #f36;
```

```
}
```

```
.box {
```

```
display: -moz-box;
```

```
display: -webkit-box;
```

```
display: box;
```

```
}
```

```
.box-horizontal {
```

```
-moz-box-orient: horizontal;
```

```
-webkit-box-orient: horizontal;
```

```
box-orient: horizontal;
```

```
}
```

```
.box-vertical {
```

```
-moz-box-orient: vertical;
```

```
-webkit-box-orient: vertical;
```

```
box-orient: vertical;
```

```
}
```

```
.box-inline-axis {
```

```
-moz-box-orient: inline-axis;
```

```
-webkit-box-orient: inline-axis;
```

```
box-orient: inline-axis;
```

```
}
```

```
.box-block-axis {
```

```
-moz-box-orient: block-axis;
```

```
-webkit-box-orient: block-axis;
```

```
box-orient: block-axis;
```

```
}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<div class="box box-horizontal">
```

```
<div>A</div>
```

```
<div>B</div>
```

```
<div>C</div>
```

```
<div>D</div>
```

```
</div>
```

```
<div class="box box-vertical">
```

```
<div>A</div>
```

```
<div>B</div>
```

```
<div>C</div>
```

```
<div>D</div>
```

```
</div>
```

```
<div class="box box-inline-axis">
```

```

<div>A</div>
<div>B</div>
<div>C</div>
<div>D</div>
</div>
<div class="box box-block-axis">
  <div>A</div>
  <div>B</div>
  <div>C</div>
  <div>D</div>
</div>
</body>
</html>

```

其效果如图 8-3 所示。

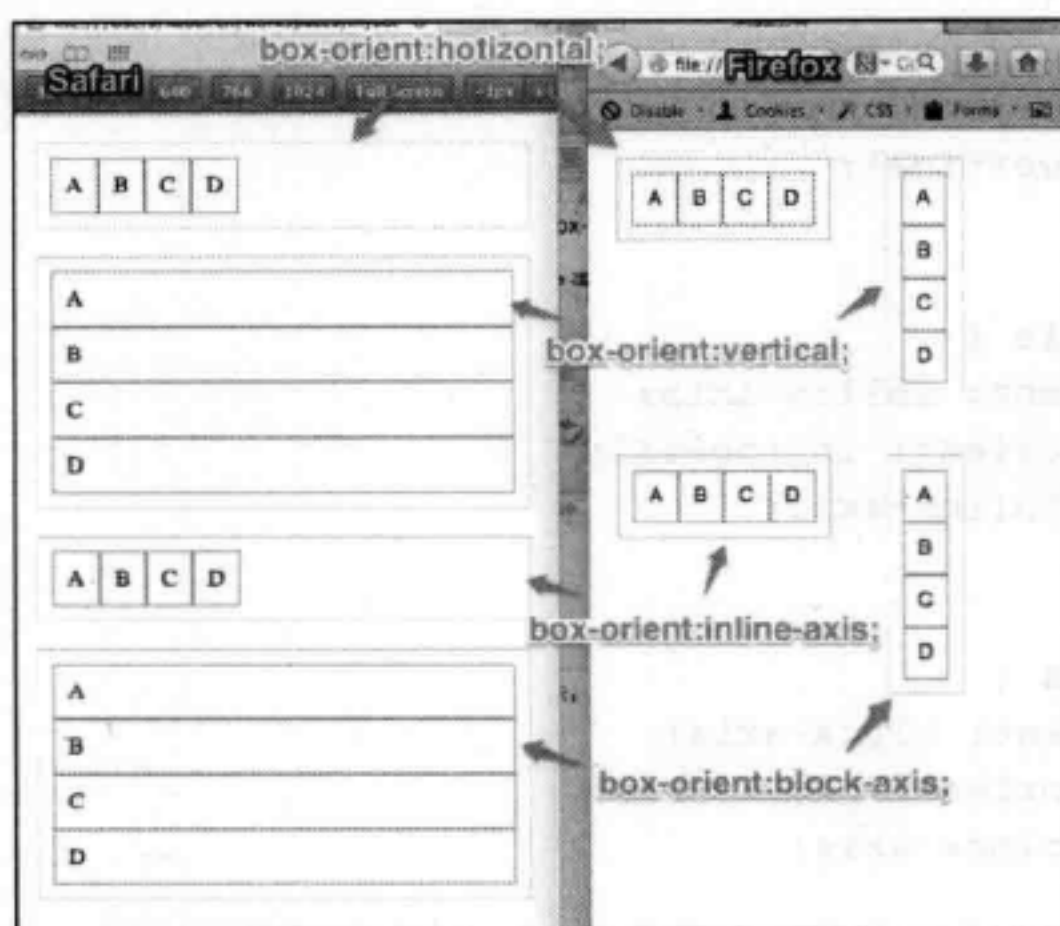


图 8-3 box-orient 在 Safari 和 Firefox 下的效果

如图 8-3 所示看出，给伸缩容器设置 horizontal 和 inline-axis 值后，可以修改文档流的默认显示方式，也就是在垂直方向从上向下显示；而给伸缩容器设置 vertical 和 block-axis 值，其显示的方式和文档流默认显示的方式一样。

其实还可以通过设置书写模式的方向，可以进一步的改变文档流的显示方向。一旦给伸缩容器显式设置了伸缩流方向后，其中 horizontal 和 inline-axis 修改的文档流显示方向，在表面上与传统的 display:inline-block 或者 display:inline 渲染的效果一样。虽然在浏览器下渲染的设计效果一样，但实际上显示的技术却完全不同。

8.2.3 布局顺序 box-direction

布局顺序是指伸缩项目在伸缩容器中的流动顺序，根据 W3C 规范可知，文档流的方向是按照文档在 HTML (DOM) 中出现的先后顺序决定的。在伸缩容器中，可以通过 box-

direction 属性来设置伸缩容器中的伸缩项目的流动顺序。

1. box-direction 属性的语法及参数

box-orient 主要是用来设置伸缩流方向，而 box-direction 属性主要是用来设置伸缩项目在伸缩容器中的流动顺序，该属性的基本语法也非常简单。

```
box-direction: normal | reverse
```

box-direction 主要包括两个属性参数，其取值简单说明如下。

- normal：正常显示顺序。如果伸缩容器设置 box-orient 的值为 horizontal 或者 inline-axis 时，伸缩项目从主轴起始开始按文档流结构顺序，从左向右按顺序排列。如果伸缩容器设置 box-orient 值为 vertical 或者 block-axis 时，伸缩项目从主轴起始点开始按文档流结构，从上到下按顺序排列。
- reverse：反向显示。如果伸缩容器设置 box-orient 的值为 horizontal 或者 inline-axis 时，伸缩项目从主轴终点开始按文档流结构的反方向，从右向左排列。如果伸缩容器设置 box-orient 值为 vertical 或者 block-axis 时，伸缩项目从主轴终点开始按文档流结构反向，从下往上排列。

可以简单理解为，normal 为默认布局顺序，按照文档结构顺序流显示，reverse 刚好与 normal 值相反，把文档结构顺序反向显示。当然其中显示顺序还和文本书写模式很有关系。

如果文本书写模式为 ltr 时，box-direction 值为 normal 时，伸缩项目从左往右显示（前提是 box-orient 值设置为 horizontal 或者 inline-axis）；如果文本书写模式为 rtl 时，box-direction 值为 normal 时，伸缩项目从右往左显示（前提是 box-orient 值设置为 horizontal 或者 inline-axis）。这个时候 box-direction 取值为 reverse 时，效果刚好与上述效果相反。同样，垂直方向的书写模式也会直接影响伸缩布局的显示顺序。

2. box-direction 属性的基本使用

文档流正常显示往往不能满足我们的需求，这时 box-direction 就能发挥优势。例如，将伸缩容器的伸缩项目反过来显示，可以直接使用 reverse。接下来通过一个简单的实例说明。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title> 伸缩布局顺序 </title>
  <style type="text/css" media="screen">
    *{
      margin: 0;
      padding: 0;
    }
    body>div {
      border: 1px solid #ccc;
      margin: 20px;
```

```

}
div > div {
  border: 1px solid #f36;
  padding: 10px;

```

```

}
.box {
  display: -moz-box;
  display: -webkit-box;
  display: box;
}

```

```

.box-horizontal {
  -moz-box-orient: horizontal;
  -webkit-box-orient: horizontal;
  box-orient: horizontal;
  width: 250px;
}

```

```

.box-vertical {
  -moz-box-orient: vertical;
  -webkit-box-orient: vertical;
  box-orient: vertical;
  height: 250px;
}

```

```

.box-direction-reverse {
  -moz-box-direction: reverse;
  -webkit-box-direction: reverse;
  box-direction: reverse;
}

```

```

</style>

```

```

</head>

```

```

<body>

```

```

  <div class="box box-horizontal">

```

```

    <div>A</div>

```

```

    <div>B</div>

```

```

    <div>C</div>

```

```

    <div>D</div>

```

```

  </div>

```

```

  <div class="box box-horizontal box-direction-reverse">

```

```

    <div>A</div>

```

```

    <div>B</div>

```

```

    <div>C</div>

```

```

    <div>D</div>

```

```

  </div>

```

```

  <div class="box box-vertical">

```

```

    <div>A</div>

```

```

<div>B</div>
<div>C</div>
<div>D</div>
</div>

<div class="box box-vertical box-direction-reverse">
  <div>A</div>
  <div>B</div>
  <div>C</div>
  <div>D</div>
</div>
</body>
</html>

```

效果如图 8-4 所示。

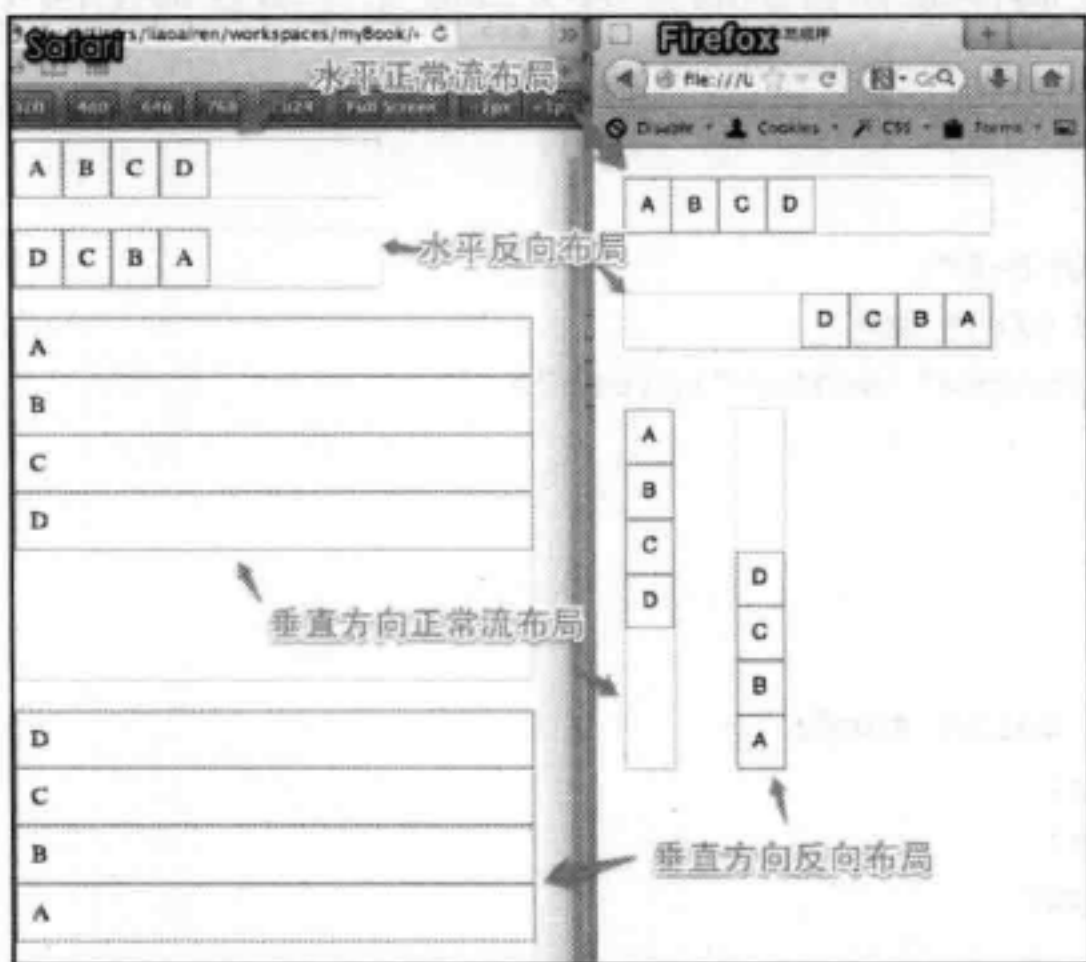


图 8-4 box-direction 在 Safari 和 Firefox 下的效果

8.2.4 伸缩换行 box-lines

在传统盒模型布局中常有元素溢出容器，在 Flexbox 模型中同样会有这样的现象发生，伸缩项目会跑出伸缩容器。为了让伸缩项目不溢出伸缩容器，可以像传统的盒模型一样，通过 overflow 属性来控制伸缩项目溢出的部分隐藏，或者通过给伸缩容器添加滚动条。不过在 Flexbox 模型中，还可以使用伸缩换行属性 box-lines 来控制伸缩项目是否溢出伸缩容器。

1. box-lines 属性的语法及参数

box-lines 用来设置伸缩容器的伸缩项目是单行还是多行显示，以及决定侧轴的方向。默认情况下都是单行或单列显示。该属性的基本语法如下。

```
box-lines: single | multiple
```


伸缩行换行属性只包括两个参数，其取值简单说明。

- ❑ **single**：伸缩容器的所有伸缩项目一行或一列显示。如果伸缩容器设置了 **overflow** 属性，可以直接控制伸缩项目是否隐藏、裁剪或者出现滚动条。
- ❑ **multiple**：指定伸缩容器多行或多列显示伸缩项目，当伸缩容器没有足够空间放置所有伸缩项目的时候，伸缩项目就会自动换行或多列显示。

如果没有给伸缩容器显式设置 **box-lines** 属性值时，一旦伸缩容器没有足够的空间容纳伸缩项目时，伸缩项目就会溢出伸缩容器。

2. **box-lines** 属性的基本使用

下面通过一个简单的实例来说明 **box-lines** 的基本使用。在例中创建了两个伸缩容器，一个伸缩容器里的所有伸缩项目水平显示，另一个伸缩容器的项目垂直显示。接下来主要通过 **box-lines** 属性来控制伸缩容器里的伸缩项目避免伸缩容器空间不足时溢出伸缩容器。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title> 伸缩行换行 </title>
  <style type="text/css" media="screen">
    *{
      margin: 0;
      padding: 0;
    }
    body>div {
      border: 1px solid #ccc;
      margin: 10px;
      width: 200px;
      height: 200px;
    }
    div > div {
      border: 1px solid #f36;
      padding: 10px;
    }
    .box {
      display: -moz-box;
      display: -webkit-box;
      display: box;
      -moz-box-pack: start;
      -webkit-box-pack: start;
      box-pack: start;
      -moz-box-align: start;
      -webkit-box-align: start;
      box-align: start;
      box-pack: start;
    }
  </style>
</head>
<body>
  <div>
    <div class="box">
      <div>伸缩项目1</div>
      <div>伸缩项目2</div>
      <div>伸缩项目3</div>
      <div>伸缩项目4</div>
      <div>伸缩项目5</div>
      <div>伸缩项目6</div>
      <div>伸缩项目7</div>
      <div>伸缩项目8</div>
      <div>伸缩项目9</div>
      <div>伸缩项目10</div>
    </div>
  </div>
</body>
</html>
```

```

-moz-box-lines: multiple;
-webkit-box-lines: multiple;
box-lines: multiple;
}

.box-horizontal {
-moz-box-orient: horizontal;
-webkit-box-orient: horizontal;
box-orient: horizontal;
}

.box-vertical {
-moz-box-orient: vertical;
-webkit-box-orient: vertical;
box-orient: vertical;
}
</style>
</head>
<body>
<div class="box box-horizontal">
<div>A</div>
<div>B</div>
<div>C</div>
<div>D</div>
<div>E</div>
<div>F</div>
<div>G</div>
<div>H</div>
<div>I</div>
<div>J</div>
<div>K</div>
</div>

<div class="box box-vertical">
<div>A</div>
<div>B</div>
<div>C</div>
<div>D</div>
<div>E</div>
<div>F</div>
<div>G</div>
<div>H</div>
<div>I</div>
<div>J</div>
<div>K</div>
</div>
</body>
</html>

```

效果如图 8-5 所示, box-lines 到目前为止还没有浏览器支持这个属性, 并且在不同的浏览内核浏览器解析也并不相同, 所以导致 box-lines 属性还无法直接使用到实际项目中。

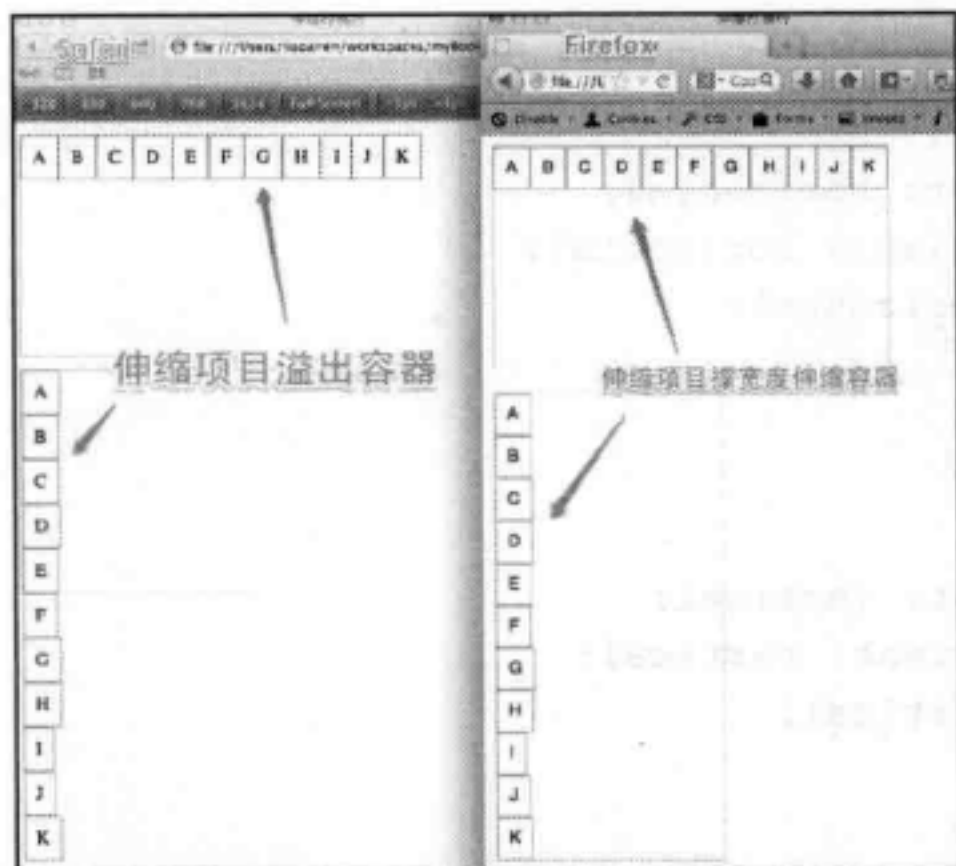


图 8-5 box-lines 在 Safari 和 Firefox 下的效果

8.2.5 主轴对齐 box-pack

主轴对齐用来设置伸缩器当前伸缩项目在主轴方向的对齐方式。指定如何在伸缩项目之间分布伸缩容器额外空间。当一行上的所有伸缩项目不能伸缩或可伸缩但是已达到最大长度时, 这一属性才会对伸缩容器额外空间进行分配。当伸缩项目溢出某一行时, 这一属性也会在项目的对齐上施加一些控制。

1. box-pack 属性的语法及参数

box-pack 属性可以在主轴方向上对伸缩容器的额外空间进行管理, 该属性的基本语法如下。

```
box-pack: start | end | center | justify
```

box-pack 主要有四个属性值, 其取值简单说明。

- ❑ start: 伸缩项目向一行的起始位置靠齐。伸缩容器沿着布局轴方向的所有额外空间都被置于布局轴的末尾。
- ❑ end: 和 start 值相反, 伸缩项目向一行的结束位置靠齐。伸缩容器沿着布局轴方向的所有额外空间都被置于布局轴的开始。
- ❑ center: 伸缩项目向一行的中间位置靠齐。伸缩容器所有额外空间平均分布在第一伸缩项目前面和最后一个伸缩项目的后面。
- ❑ justify: 伸缩项目会平均分布在一行里。伸缩容器所有额外空间平均分布在所有伸缩项目之间, 而且在第一个伸缩项目之前和最后一个伸缩项目之后不分配伸缩容器的任何额外空间。

2. box-pack 属性的基本使用

box-pack 主要是用来控制伸缩项目在主轴的对齐方式, 以及管理伸缩容器额外空间。伸缩布局主轴的方向和 box-pack 的值直接影响了伸缩容器额外空间的位置。换句话说, box-orient 和 box-pack 的值可以直接管理伸缩容器额外空间。

当 box-pack 取值为 start, 第一个伸缩项目的起始边缘置于伸缩容器的主轴起点位置, 下一个伸缩项目的起始边缘与第一伸缩容器的末尾边缘边挨边一放置在一起, 其他的伸缩项目依此方式沿着布局轴继续排列。伸缩容器沿着布局轴方向的所有额外空间都置于布局轴的末尾。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>box-pack 取值为 start 的基本使用 </title>
  <style type="text/css" media="screen">
    .box {
      width: 500px;
      height: 200px;
      margin: 20px;
      border: 1px solid #ccc;
      font-size: 20px;
      font-weight: bold;
      color: #fff;
    }
    .box > div {
      padding: 10px;
    }
    .box div:nth-child(1){
      background-color: #305ed5;
      line-height: 50px;
    }
    .box div:nth-child(2){
      background-color: #e727b3;
      line-height: 80px;
    }
    .box div:nth-child(3) {
      background-color: #3f8514;
      line-height: 100px;
    }
    .box div:nth-child(4) {
      background-color: #e6b710;
    }
    .box div:nth-child(5){
      background-color: #f96;
    }
  /*box*/
  .box {
```

```

display: -webkit-box;
display: -moz-box;
display: box;
-webkit-box-align: start;
-moz-box-align: start;
box-align: start;
-webkit-box-pack: start;
-moz-box-pack: start;
box-pack: start;
}

.box-orient-vertical{
  -webkit-box-orient: vertical;
  -moz-box-orient: vertical;
  box-orient: vertical;
  height:400px;
}
</style>
</head>
<body>
  <div class="box box-pack-start">
    <div class="box-item">box1</div>
    <div class="box-item">box2</div>
    <div class="box-item">box3</div>
    <div class="box-item">box4</div>
    <div class="box-item">start</div>
  </div>
  <div class="box box-pack-start box-orient-vertical">
    <div class="box-item">box1</div>
    <div class="box-item">box2</div>
    <div class="box-item">box3</div>
    <div class="box-item">box4</div>
    <div class="box-item">start</div>
  </div>
</body>
</html>

```

其效果如图 8-6 所示。

当 box-pack 取值为 end 时, 其效果与 “start” 效果相反。第一个伸缩项目的末尾边被置于伸缩容器的结束位置; 下一个伸缩项目的末尾边缘与第一个伸缩项目的起始边缘边挨边地放置在一起; 其他伸缩项目依此方式沿着布局轴方向继续排列。伸缩容器沿着布局轴方向所有额外空间都被放置于布局轴的开始。

```

<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">

```

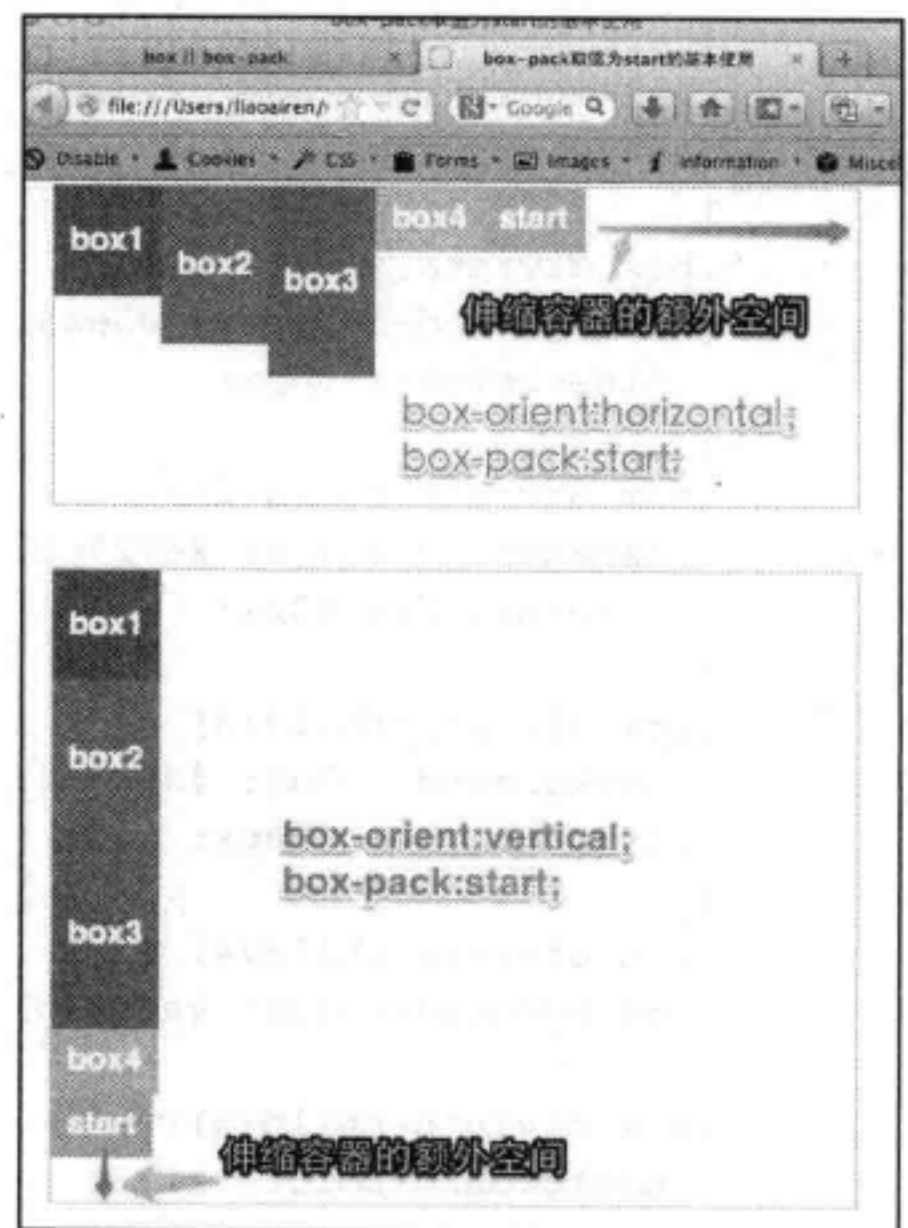


图 8-6 box-pack 取值为 start 在 Firefox 下的效果

```

<title>box-pack 取值为 start 的基本使用 </title>
<style type="text/css" media="screen">
...// 省略部分内容
.box {
    display: -webkit-box;
    display: -moz-box;
    display: box;
    -webkit-box-align: start;
    -moz-box-align: start;
    box-align: start;

    -webkit-box-pack: end;
    -moz-box-pack: end;
    box-pack: end;
}
...// 省略部分内容

```

效果如图 8-7 所示。

当 box-pack 取值为 center 时，所有伸缩项目边挨边放置在一起，如同 start 和 end 关键词的描述中所说的那样。但是，该组伸缩项目在伸缩容器的起始和末尾边缘之间位于中间位置，这样，伸缩容器所有额外的空间都平均分布在第一个伸缩项目的前面和最后一个伸缩项目的后面。

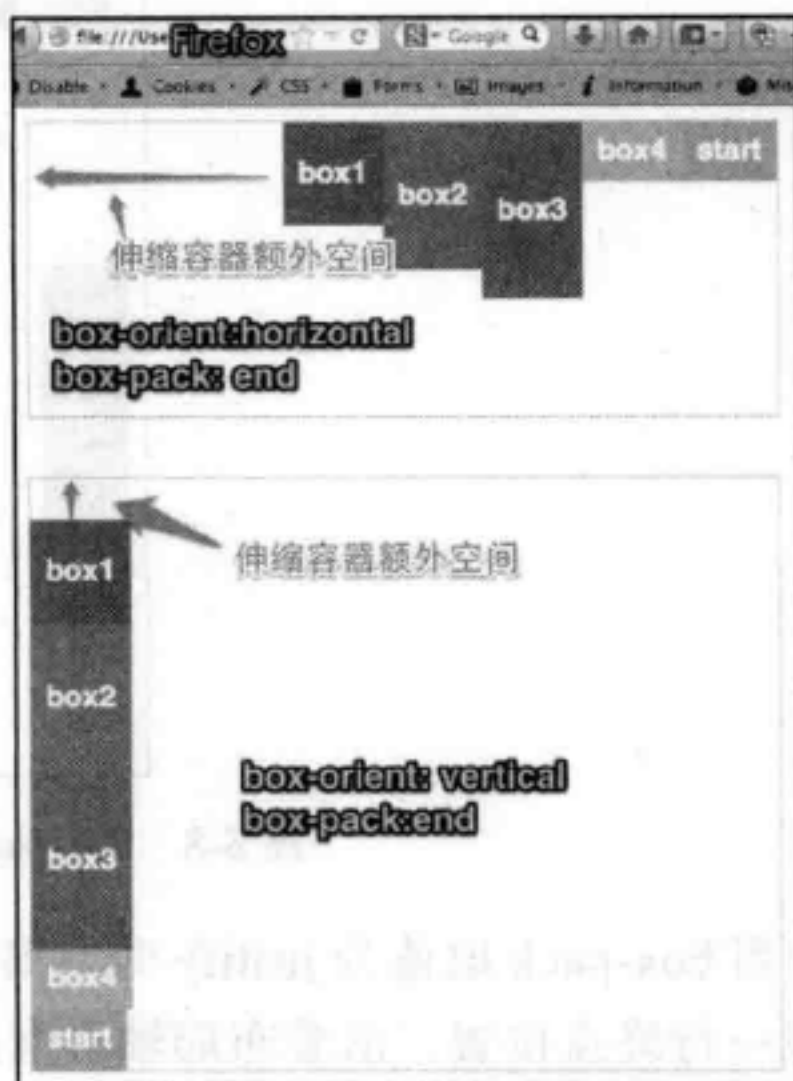


图 8-7 box-pack 取值为 end 在 Firefox 的效果

```

<!DOCTYPE HTML>
<html lang="en-US">
<head>
    <meta charset="UTF-8">
    <title>box-pack 取值为 center 的基本使用 </title>
    <style type="text/css" media="screen">
...// 省略部分内容
.box {
    display: -webkit-box;
    display: -moz-box;
    display: box;
    -webkit-box-align: start;
    -moz-box-align: start;
    box-align: start;

    -webkit-box-pack: center;
    -moz-box-pack: center;
    box-pack: center;
}
...// 省略部分内容

```

其效果如图 8-8 所示。

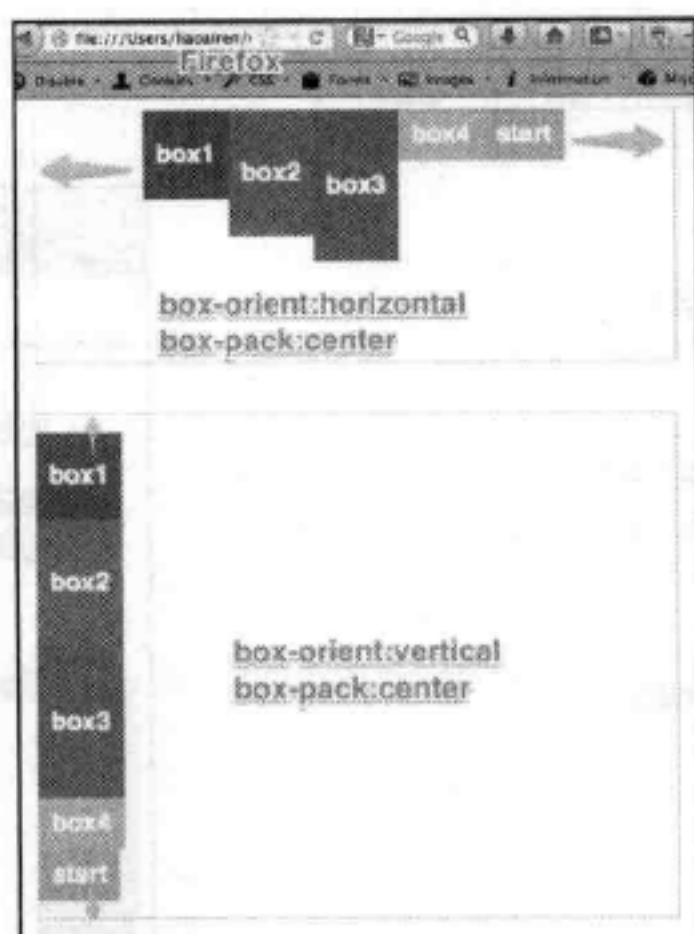


图 8-8 box-pack 取值为 center 在 Firefox 的效果

当 box-pack 取值为 justify 时，第一个伸缩项目始终在一行开始位置，最后一个伸缩项目在一行终点位置。沿着布局轴方向的任何额外空间都平均分布于各个伸缩项目之间。如果伸缩容器没有足够空间或者只有一个伸缩项目时，会以 start 方式展示。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>box-pack 取值为 justify 的基本使用 </title>
  <style type="text/css" media="screen">
...// 省略部分内容
  .box {
    display: -webkit-box;
    display: -moz-box;
    display: box;
    -webkit-box-align: start;
    -moz-box-align: start;
    box-align: start;

    -webkit-box-pack: justify;
    -moz-box-pack: justify;
    box-pack: justify;
  }
...// 省略部分内容
```

效果如图 8-9 所示。



注意

box-pack 取值为 justify 时，在 Firefox 浏览器下会以 start 方式展示。

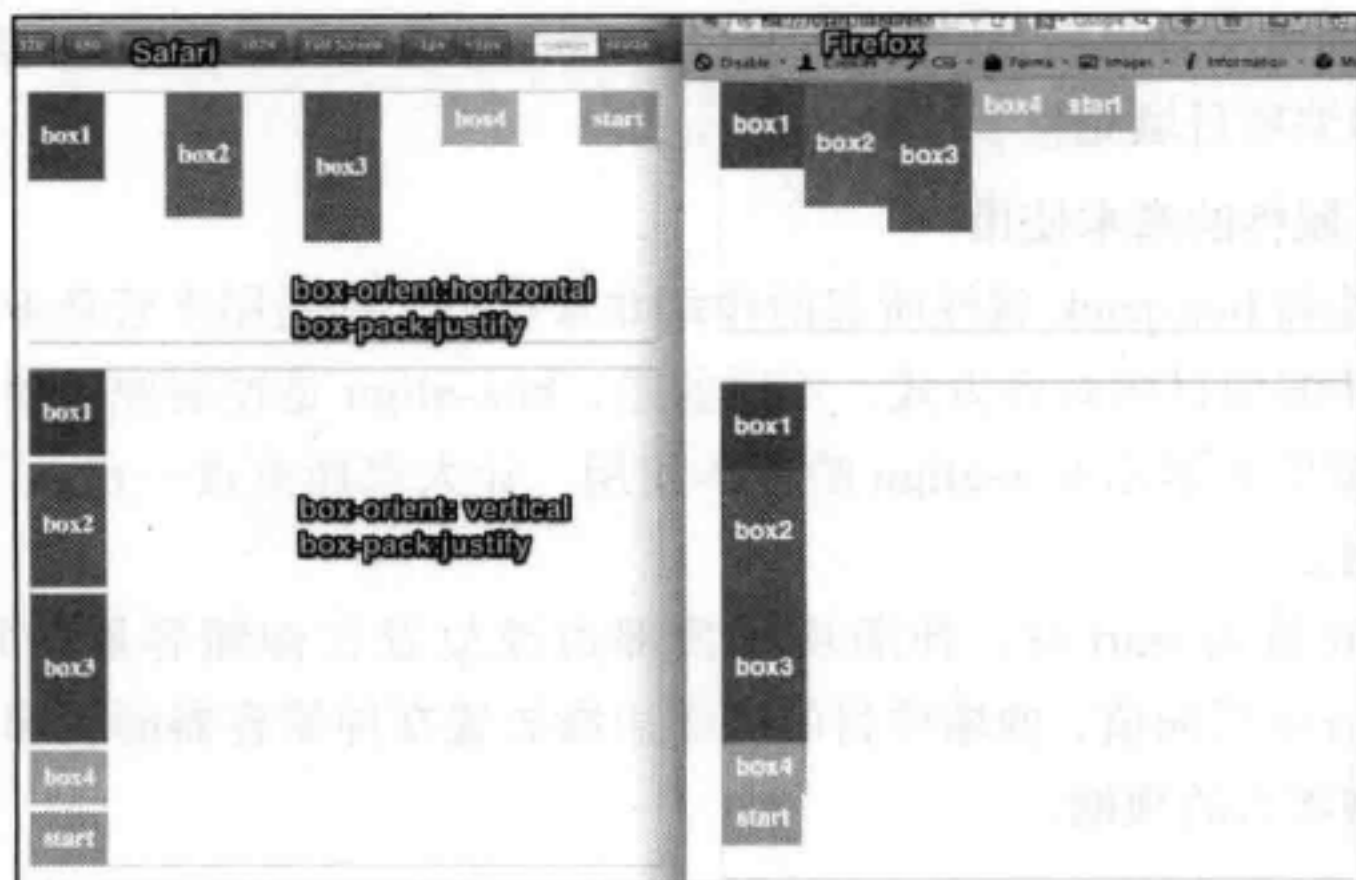


图 8-9 box-pack 取值为 justify 在 Safari 和 Firefox 下的效果

box-pack 除了和伸缩流方向有关之外，当 box-pack 取值为 start 和 end 时，还与伸缩布局顺序 box-direction 有关。当 box-pack 取值为 start，box-direction 取值为 reverse 时，box-pack 的 start 和 end 效果等同。换过来，当 box-pack 取值为 end，box-direction 取值为 reverse 时，box-pack 的 end 与 start 效果等同。

8.2.6 侧轴对齐 box-align

box-align 属性和 box-pack 同样是用来管理伸缩容器额外空间，不同的是，box-pack 是用来管理伸缩容器主轴方向的额外空间，而 box-align 是用来管理伸缩容器侧轴方向的额外空间，也就是垂直于主轴方向。换句话说就是用来定义伸缩项目在伸缩容器的当前行的侧轴上对齐方式。

1. box-align 属性的语法及参数

box-align 主要是用来管理伸缩容器在侧轴方向的额外空间，也就是用来定义伸缩项目在侧轴的对齐方式。该属性的语法如下。

```
box-align: start | end | center | baseline | stretch
```

box-align 属性参数在 box-pack 的基础上减去了两端对齐 justify，但增加了另外两个属性参数 baseline 和 stretch。各属性参数简单介绍如下。

- ❑ start: 伸缩项目顶部边缘放置在伸缩容器的顶端，伸缩容器的额外空间放置在伸缩项目底端。
- ❑ end: 与 start 值相反，所有伸缩项目底部边缘放置在伸缩容器底端，伸缩容器额外空间放置在伸缩项目顶部。
- ❑ center: 伸缩项目紧靠在一起，垂直居中于伸缩容器。伸缩容器额外的空间平均分配在伸缩项目的顶部和底部。

- baseline: 伸缩项目根据它们的基线对齐。伸缩容器额外空间可前可后显示。
- stretch: 伸缩项目填充整个伸缩容器。

2. box-align 属性的基本使用

box-align 属性和 box-pack 属性所起的作用非常类似，都是用来管理伸缩容器额外的空间，也就是定义伸缩项目的对齐方式，不同的是，box-align 是控制伸缩项目在伸缩容器侧轴的对齐方式。接下来演示 box-align 的基本使用，让大家能更进一步地了解 box-align 各属性值所起的作用。

当 box-align 取值为 start 时，伸缩项目顶部边缘放置在伸缩容器的顶端，如果 box-direction 设置 reverse 反向值，伸缩项目的底部边缘放置在伸缩容器的底部，伸缩容器额外的空间放置在伸缩项目的顶端。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>box-align 取值为 start 的基本使用 </title>
  <style type="text/css" media="screen">
    .box {
      width: 330px;
      height: 200px;
      margin: 20px;
      border: 1px solid #ccc;
      font-size: 20px;
      font-weight: bold;
      color: #fff;
    }
    .box > div {
      padding: 10px;
    }
    .box div:nth-child(1){
      background-color: #305ed5;
      line-height: 50px;
    }
    .box div:nth-child(2){
      background-color: #e727b3;
      line-height: 80px;
    }
    .box div:nth-child(3) {
      background-color: #3f8514;
      line-height: 100px;
    }
    .box div:nth-child(4) {
      background-color: #e6b710;
    }
    .box div:nth-child(5){
      background-color: #f96;
```



```

}
/*box*/
.box {
  display: -webkit-box;
  display: -moz-box;
  display: box;
  -webkit-box-align: start;
  -moz-box-align: start;
  box-align: start;
}
.box-orient-vertical{
  -webkit-box-orient: vertical;
  -moz-box-orient: vertical;
  box-orient: vertical;
  height:400px;
}
</style>
</head>
<body>
  <div class="box box-pack-start">
    <div class="box-item">box1</div>
    <div class="box-item">box2</div>
    <div class="box-item">box3</div>
    <div class="box-item">box4</div>
    <div class="box-item">start</div>
  </div>
  <div class="box box-pack-start box-orient-vertical">
    <div class="box-item">box1</div>
    <div class="box-item">box2</div>
    <div class="box-item">box3</div>
    <div class="box-item">box4</div>
    <div class="box-item">start</div>
  </div>
</body>
</html>

```

其效果如图 8-10 所示。

当 box-align 取值为 end 时,效果与取值为 start 刚好相反。所有伸缩项目底端边缘都放置在伸缩容器底端,伸缩容器的额外空间在所有伸缩项目的顶端。如果伸缩容器设置 box-direction 为 reverse 值时,所有伸缩项目的顶端边缘都放置在伸缩容器的顶部,伸缩容器的额外空间在所有伸缩项目的底部。

/* 在上例基础上调整 .box 样式 */

```

.box {
  display: -webkit-box;
  display: -moz-box;

```

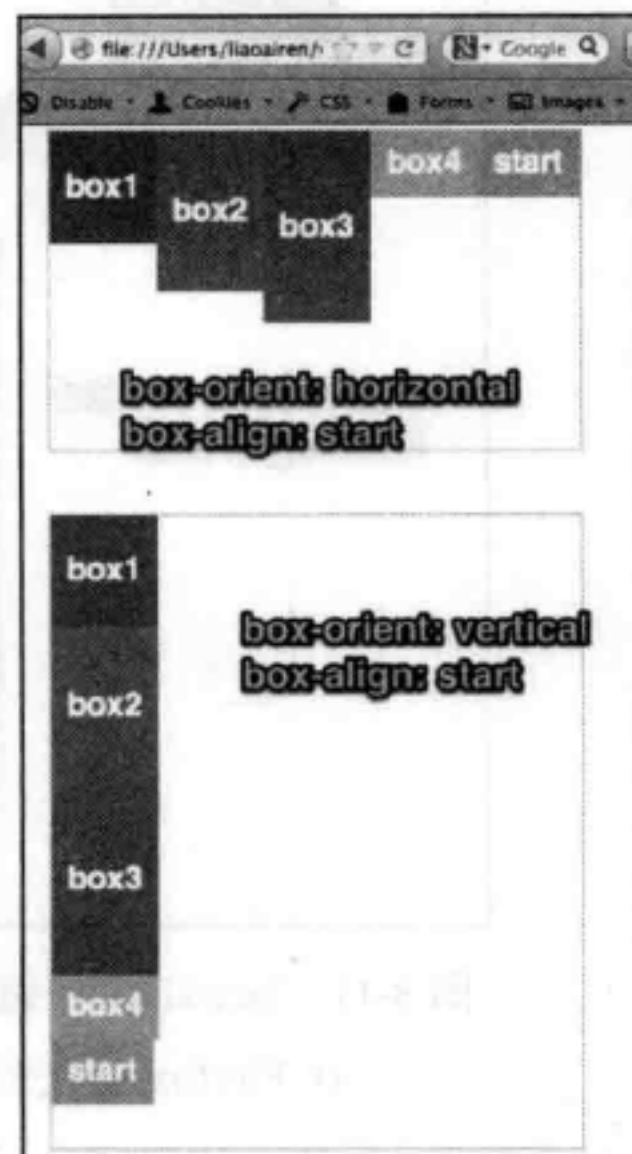


图 8-10 box-align 取值为 start 在 Firefox 的效果

```

display: box;
-webkit-box-align: end;
-moz-box-align: end;
box-align: end;
}

```

效果如图 8-11 所示。

当 `box-align` 取值为 `center` 时，所有伸缩项目垂直居中，伸缩容器额外的空间均匀地分配在伸缩项目的顶部和底部。常用来制作元素的垂直居中效果。

```

/* 在上例基础上调整 .box 样式 */
.box {
display: -webkit-box;
display: -moz-box;
display: box;
-webkit-box-align: center;
-moz-box-align: center;
box-align: center;
}

```

效果如图 8-12 所示。

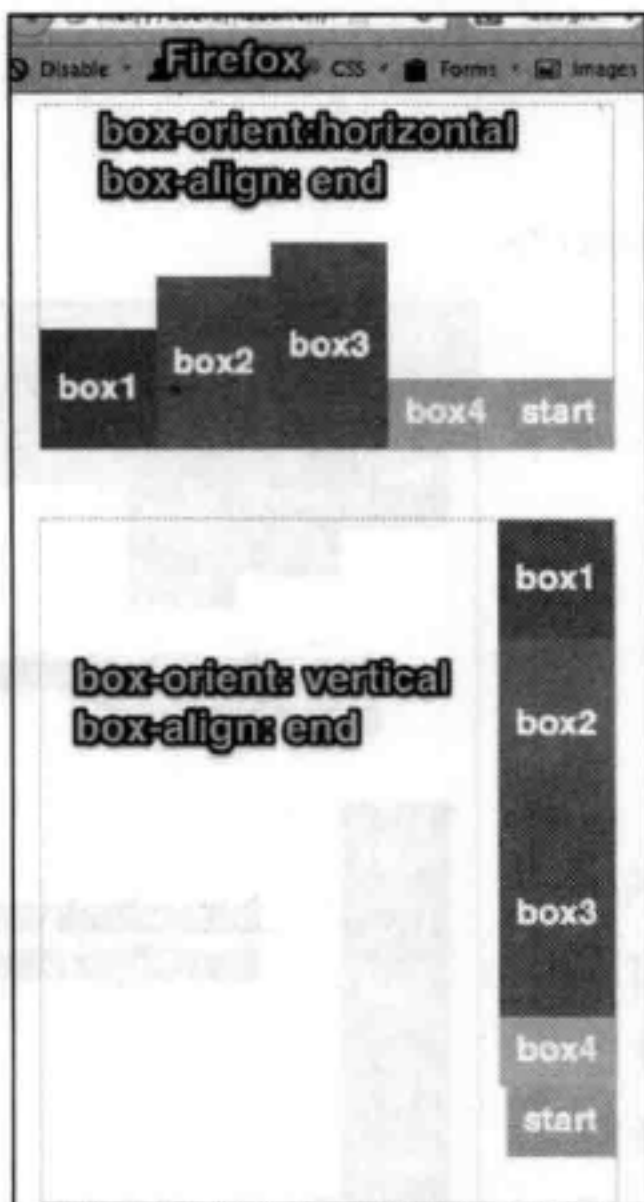


图 8-11 `box-align` 取值为 `end`
在 Firefox 的效果

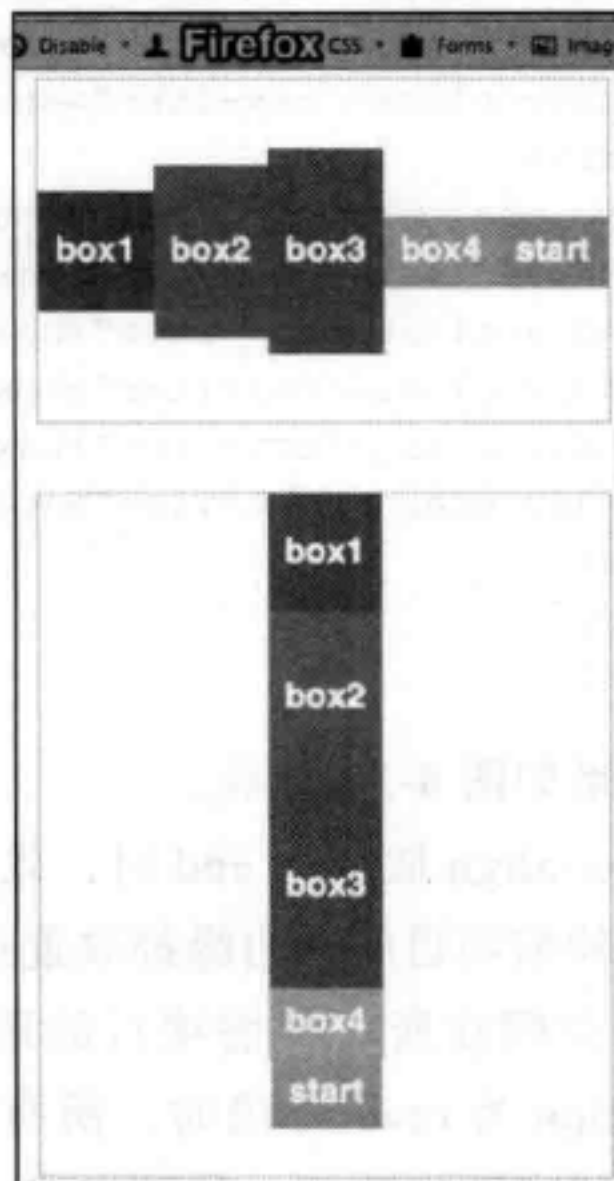


图 8-12 `box-align` 取值为 `center`
在 Firefox 效果

当伸缩容器设置 `box-direction` 取值为 `reverse` 时，并不会影响 `box-align` 取值为 `center` 的效果。

当 `box-align` 取值为 `baseline` 时, 伸缩项目根据它们的基线对齐。所有伸缩项目基线(取决于 `box-direction` 属性的起始边缘和末尾边缘)都彼此对齐。占用空间最多且垂直于布局轴的伸缩项目遵循 `start` 规则; 然后所有剩余伸缩项目的基线与该伸缩项目的基线对齐。

```
/* 在上例基础上调整 .box 样式 */
.box {
    display: -webkit-box;
    display: -moz-box;
    display: box;
    -webkit-box-align: baseline;
    -moz-box-align: baseline;
    box-align: baseline;
}
```

效果如图 8-13 所示。

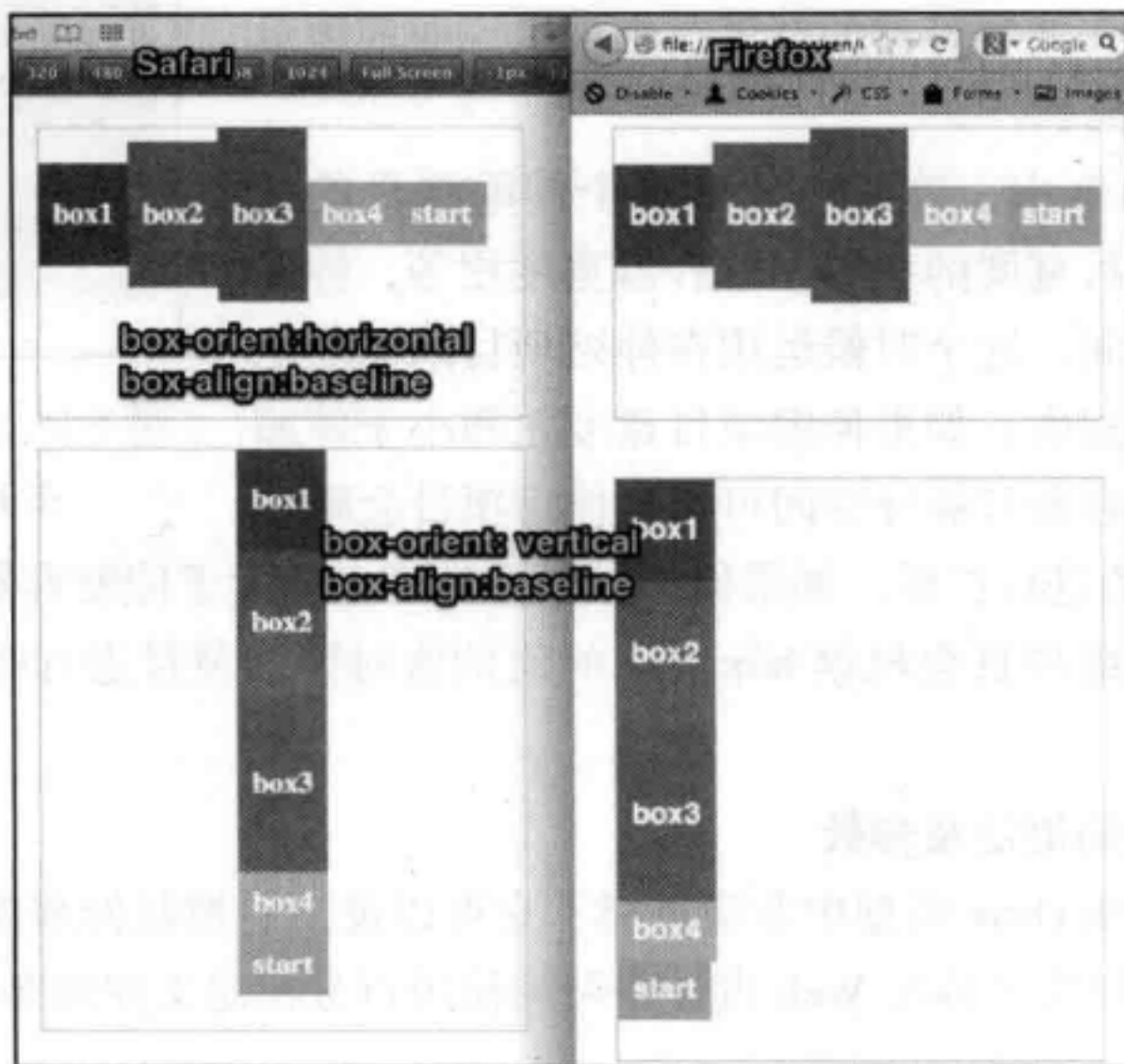


图 8-13 `box-align` 取值为 `baseline` 在 Safari 和 Firefox 的效果

`box-align` 取值为 `baseline` 时, 而且 `box-orient` 值为 `vertical` 时, 在 Webkit 内核浏览器中会以 `center` 方式展示, 而在 Firefox 中却以 `start` 方式展示。

当 `box-align` 取值为 `stretch` 时, 伸缩项目拉伸填充整个伸缩容器。此值会使用伸缩项目的外边距盒的尺寸在遵照 `min/max-width` 和 `min/max-height` 属性的限制下尽可能接近所在行的尺寸。将对每个伸缩项目进行拉伸以便完全填充垂直于布局的可用空间。如果设置伸缩项目的 `max-width/height` 属性优先并且遵循 `start` 规则。只要有一个伸缩项目设置 `min/max-height` 或 `height`, 伸缩项目都会以 `start` 规则对齐。

/* 在上例基础上调整 .box 样式 */

```
.box {
    display: -webkit-box;
    display: -moz-box;
    display: box;
    -webkit-box-align: stretch;
    -moz-box-align: stretch;
    box-align: stretch;
}
```

其效果如图 8-14 所示。

8.2.7 伸缩性 box-flex

box-flex 属性能够灵活地控制伸缩项目在伸缩容器中的显示空间。注意，显示空间包括了伸缩项目的宽度和高度，而不只是伸缩项目所在伸缩容器的宽度，也可以说伸缩项目在伸缩容器中所占的面积。

在一个水平导向框中，首先会计算每个伸缩项目的宽度。如果所有伸缩项目宽度的和与伸缩容器宽度相等，伸缩容器就没有额外的空间，这个时候运用在伸缩项目的宽度是每个伸缩项目的其次宽度；如果伸缩项目宽度之和小于伸缩容器宽度，这时伸缩容器有额外空间可用，伸缩项目会根据 box-flex 对应的比例值进行扩展。如果伸缩项目的宽度之和大于伸缩容器宽度，伸缩容器没有足够空间，这时伸缩项目会根据 box-flex 的比例值对伸缩项目进行收缩，以防止伸缩项目溢出伸缩容器。

1. box-flex 属性的语法及参数

box-flex 属性在 Flexbox 模型中非常重要，它可以灵活地控制伸缩项目的宽度来填充伸缩容器额外的空间。解决了传统 Web 设计中习惯使用百分比定义伸缩布局的弊端。box-flex 属性的基本语法如下。

```
box-flex: <number>
```

其属性值就是一个整数或者小数。当伸缩容器中包含了多个定义了 box-flex 属性的伸缩项目时，浏览器将会把这些伸缩项目的 box-flex 值相加，然后浏览器会根据它们各自的值占总值的比例来分配伸缩容器额外的空间给各个伸缩项目。

2. box-flex 的基本使用

box-flex 属性实际上不能阻止伸缩项目保持固有大小，但是它会强制伸缩项目内部的内容自动适应而不会把伸缩项目本身撑大。这样一来，让伸缩项目能够伸缩性适应伸缩容器的宽度。如果伸缩项目的固有宽度之和小于伸缩容器的宽度，伸缩项目会自动增加宽度以

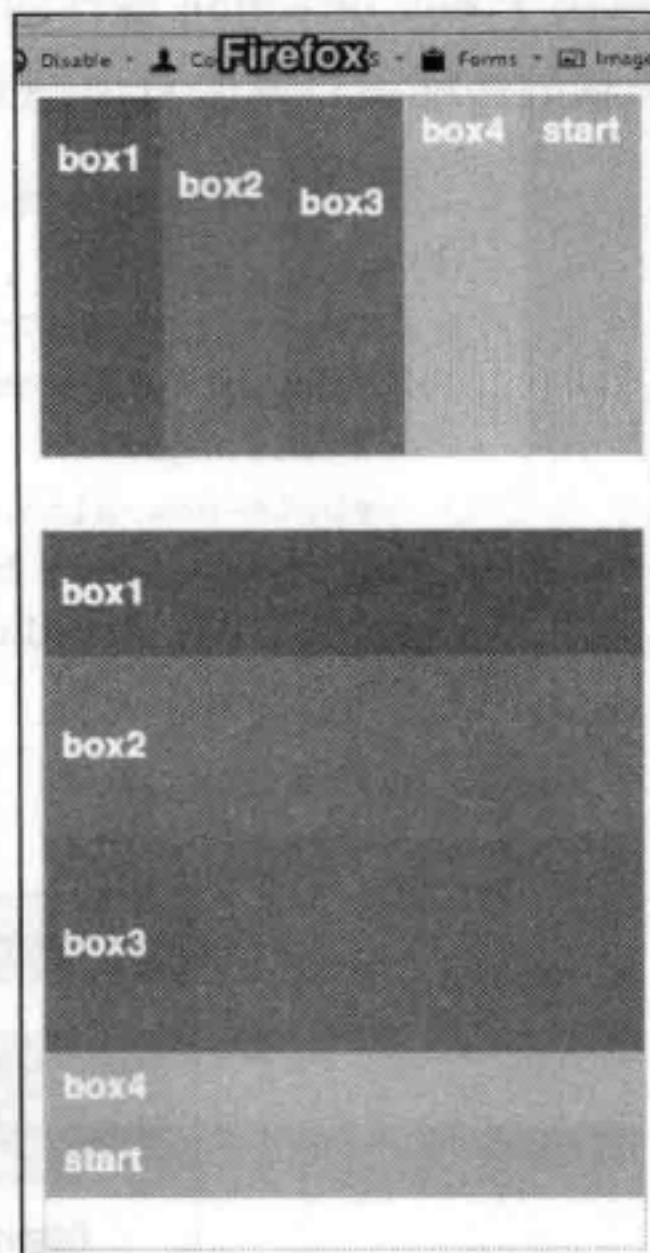


图 8-14 box-align 取值 stretch 在 Firefox 下的效果

填充伸缩容器。反之，伸缩项目的固有的宽度之和大于伸缩容器的宽度，伸缩项目会自动减少宽度以适应伸缩容器，不让伸缩项目溢出伸缩容器。

伸缩项目具体增加或者减小的宽度值取决于伸缩项目的固有宽度的比例，而不是某个绝对的数值。设置了 box-flex 值的伸缩项目是按照特定比例来弹性适应的。举个例子，一个伸缩项目设置 box-flex 值为 2，另一个伸缩项目设置 box-flex 值为 1，伸缩容器额外的宽度将分成三等分，设置 box-flex 为 2 的伸缩项目会占两份，而设置 box-flex 为 1 的伸缩项目会占一份。

下面通过一个简单的实例来说明这一切。有一个灰色的伸缩容器 flex-box，其宽度是 960px，里面包含了两个伸缩项目，而且伸缩项目的宽度由其内部的文本决定，其中绿色的那个伸缩项目宽度为 240px，另一个橙色项目的伸缩宽度为 510px。这样伸缩容器还有 210px 是额外的空间。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>box-flex 的基本使用 </title>
  <style type="text/css" media="screen">
    *{
      margin: 0;
      padding: 0;
    }
    body {
      color: #fff;
      font-size: 20px;
    }
    .box-flex {
      width: 960px;
      background: gray;
      margin: 100px;
      display: -webkit-box;
      display: -moz-box;
      display: box;
    }
    .box-flex p:first-child{
      background: green;
    }
    .box-flex p:last-child{
      background: orange;
    }
  </style>
</head>
<body>
  <div class="box-flex">
    <p> 我是一个绿色的伸缩项目。</p>
    <p> 我是一个橙色的伸缩项目，我是用来测试 box-flex 的使用。</p>
```

```

</div>
</body>
</html>

```

初步效果如图 8-15 所示。

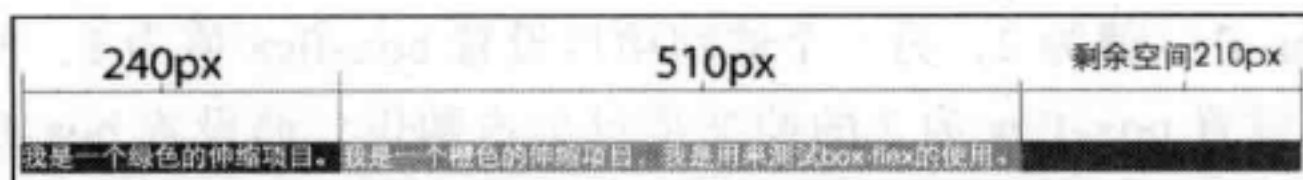


图 8-15 绿色和橙色宽度由内容决定，伸缩容器额外空间还有 210px

把两个伸缩项目都设置 box-flex 值为 1，就会把伸缩容器额外的空间 210px 平均分成两份，添加到每个伸缩项目上，这个时候绿色的伸缩项目宽度不再是内容宽度，而是 345px (240+105)，橙色的伸缩项目宽度也不再是内容的宽度，而是 615px (510+105)，如下所示。

```

.box-flex p {
  -moz-box-flex: 1;
  -webkit-box-flex: 1;
  box-flex: 1;
}

```

效果如图 8-16 所示。

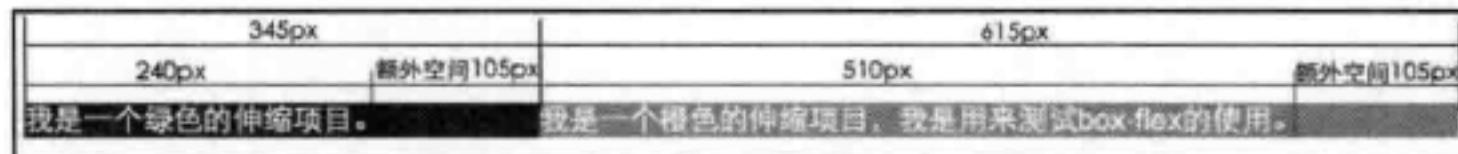


图 8-16 两个伸缩项目设置 box-flex 值为 1，增加同样额外空间 105px

在前面的基础上做一定调整，同样在绿色的伸缩项目上设置 box-flex 为 1，不过在橙色伸缩项目是设置 box-flex 为 2。此时伸缩容器的额外空间 210px 将会平均分成三份，每份宽度为 70px。这时绿色的伸缩项目获取伸缩容器额外空间为 70px，而橙色的伸缩项目获取伸缩容器额外空间为 140px。也就是说当橙色伸缩项目设置 box-flex 为 2，它分到的伸缩容器的额外空间就是绿的 2 倍，如下所示。

```

.box-flex p:first-child{
  background: green;
  -moz-box-flex: 1;
  -webkit-box-flex: 1;
  box-flex: 1;
}
.box-flex p:last-child{
  background: orange;
  -moz-box-flex: 2;
  -webkit-box-flex: 2;
  box-flex: 2;
}

```

效果如图 8-17 所示。

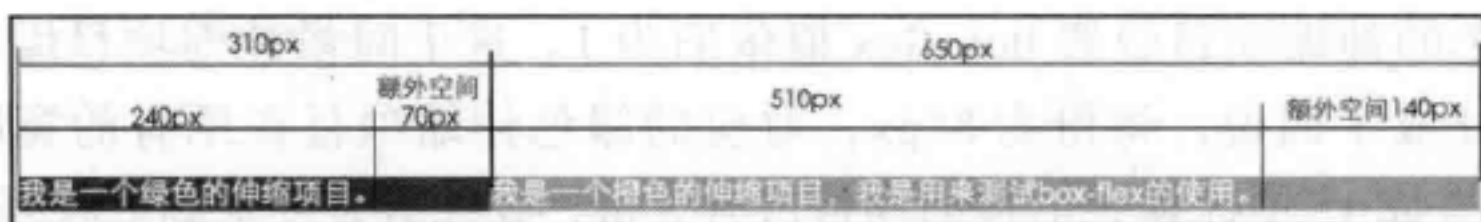


图 8-17 box-flex 的比例为 2:1 的效果

上面主要通过示例向大家介绍了伸缩项目宽度之和小于伸缩容器时，伸缩项目会根据自身设置的 box-flex 值，按比例进行扩展宽度填充伸缩容器。如果伸缩项目宽度之和大于伸缩容器，伸缩项目会根据自身设置的 box-flex 值，按比例使伸缩项目宽度适应伸缩容器，以免伸缩项目内容溢出伸缩容器。

为了能更好地说明问题，在前面的示例基础上，将伸缩容器的宽度设置 410px，这个时候伸缩项目宽度之和就大于伸缩容器的宽度。代码如下所示。

```
...// 省略部分代码
.box-flex {
    width: 410px;
    border: 1px solid red;
    background: gray;
    margin: 100px;
    display: -webkit-box;
    display: -moz-box;
    display: box;
}
.box-flex p:first-child{
    background: green;
}
.box-flex p:last-child{
    background: orange;
}
...// 省略部分代码
```

效果如图 8-18 所示。

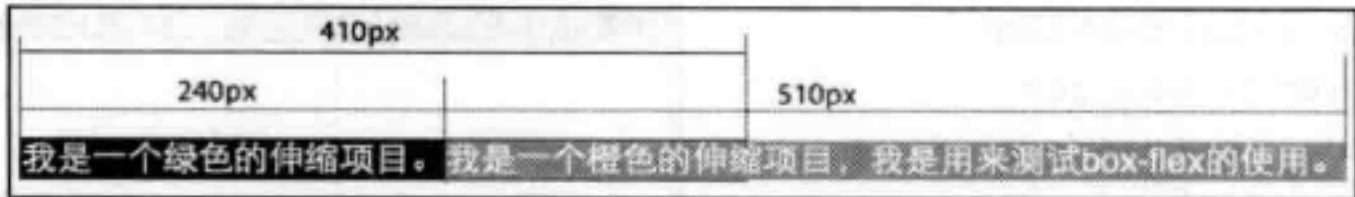


图 8-18 伸缩项目宽度之和大于伸缩容器宽度的效果

这个时候，伸缩项目溢出了伸缩容器 340px。为了让伸缩项目不溢出伸缩容器，需要在伸缩项目上设置 box-flex 属性。首先来看第一种情形，给伸缩项目设置 box-flex 的值为 1。这个时候伸缩项目会进行收缩，将溢出的宽度 340px 平均分成两份，每份 170px。对应的伸缩项目在原有的宽度上将减少 170px，也就是说，绿色伸缩项目宽度收缩后变成 70px (240-170)，橙色伸缩项目宽度收缩后变成 340px (510-170)，如图 8-19 所示。

同样可以为每个伸缩项目设置不同的伸缩比，例如，给橙色的伸缩项目设置 box-flex

值为 3，给绿色的伸缩项目设置 box-flex 值依旧为 1。这个时候伸缩项目溢出伸缩容器的 340px 宽度就分成了四份，每份为 85px，对应的绿色伸缩项目在原有的宽度基础上收缩 85px 后，宽度变成 155px；橙色的伸缩项目占有 3 份，在原有宽度基础上收缩 255px (3*85) 后，宽度变成 255px，如下所示。

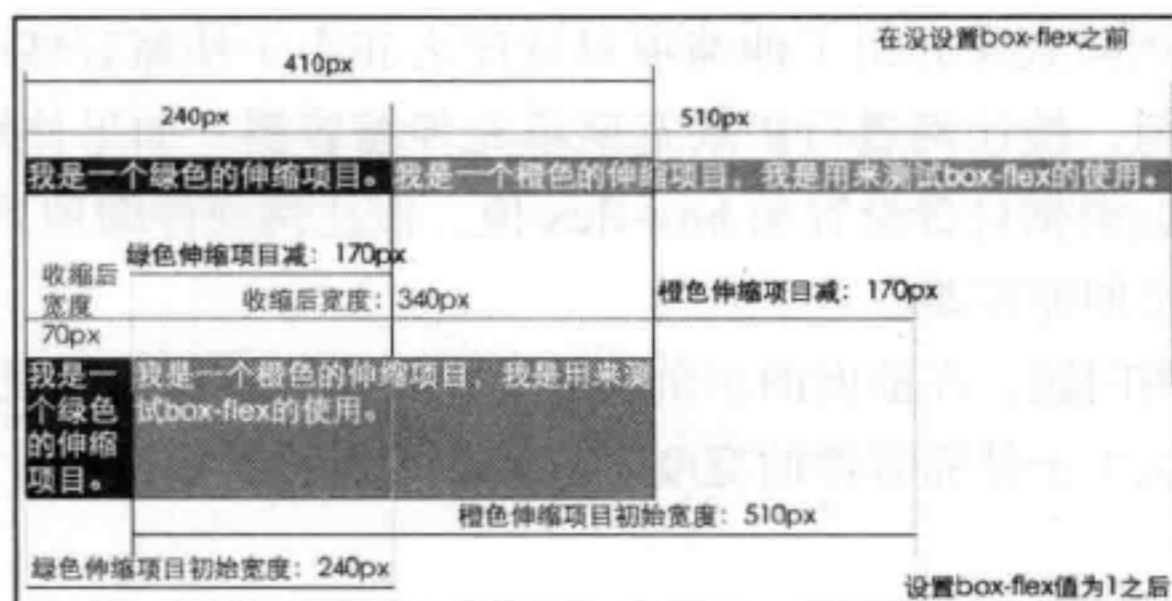


图 8-19 伸缩项目设置 box-flex 值为 1 后，伸缩项目不溢出伸缩容器的效果

...// 省略部分代码

```
.box-flex {
  width: 410px;
  border: 1px solid red;
  background: gray;
  margin: 100px;
  display: -webkit-box;
  display: -moz-box;
  display: box;
}
.box-flex p:first-child{
  background: green;
  -moz-box-flex: 1;
  -webkit-box-flex: 1;
  box-flex: 1;
}
.box-flex p:last-child{
  background: orange;
  -moz-box-flex: 3;
  -webkit-box-flex: 3;
  box-flex: 3;
}
```

...// 省略部分代码



效果如图 8-20 所示。

图 8-20 box-flex 的比例 3 : 1，伸缩项目收缩的效果

8.2.8 显示顺序 box-ordinal-group

伸缩容器中的伸缩项目默认显示顺序是遵循文档流在源码中出现的先后顺序，也就是 DOM 结构中先后顺序。在 Flexbox 模型中，伸缩项目具有其自己独立的源顺序，使用 box-

ordinal-group 属性可以修改伸缩项目在页面中的显示顺序,也可以用这个属性实现排序组。

1. box-ordinal-group 属性的语法及参数

box-ordinal-group 属性在 Flexbox 模型中的主要功能重置伸缩项目的源顺序,能够设置每个伸缩项目在伸缩容器的在页面中的具体显示位置,还可以设置伸缩项目的排序组。其语法如下所示。

```
box-ordinal-group: <integer>
```

box-ordinal-group 属性只有一个参数 <integer>, 默认值为 1。而 <integer> 为正整数,主要用来设置伸缩项目在伸缩容器的位置顺序序列号。伸缩项目的排列将根据这个属性值从小到大进行排列,值越大,伸缩项目越排在后面。

2. box-ordinal-group 属性的基本使用

在没有显式给伸缩项目设置 box-ordinal-group 值时,伸缩项目的 box-ordinal-group 默认值为 1,并且在伸缩容器中显示的顺序将按照页面文档流加载的顺序进行排列。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>box-ordinal-group 的基本使用 </title>
  <style type="text/css" media="screen">
    *{
      margin: 0;
      padding: 0;
    }
    body {
      color: #fff;
      font-size: 20px;
    }
    .box-flex {
      border: 1px solid red;
      background: gray;
      margin: 100px;
      display: -webkit-box;
      display: -moz-box;
      display: box;
    }
    .box-flex div {
      background: #f36;
      padding: 20px;
      margin: 5px;
    }
  </style>
</head>
<body>
  <div class="box-flex">
```



```

<div>Box1</div>
<div>Box2</div>
<div>Box3</div>
<div>Box4</div>
<div>Box5</div>
</div>
</body>
</html>

```

效果如图 8-21 所示。

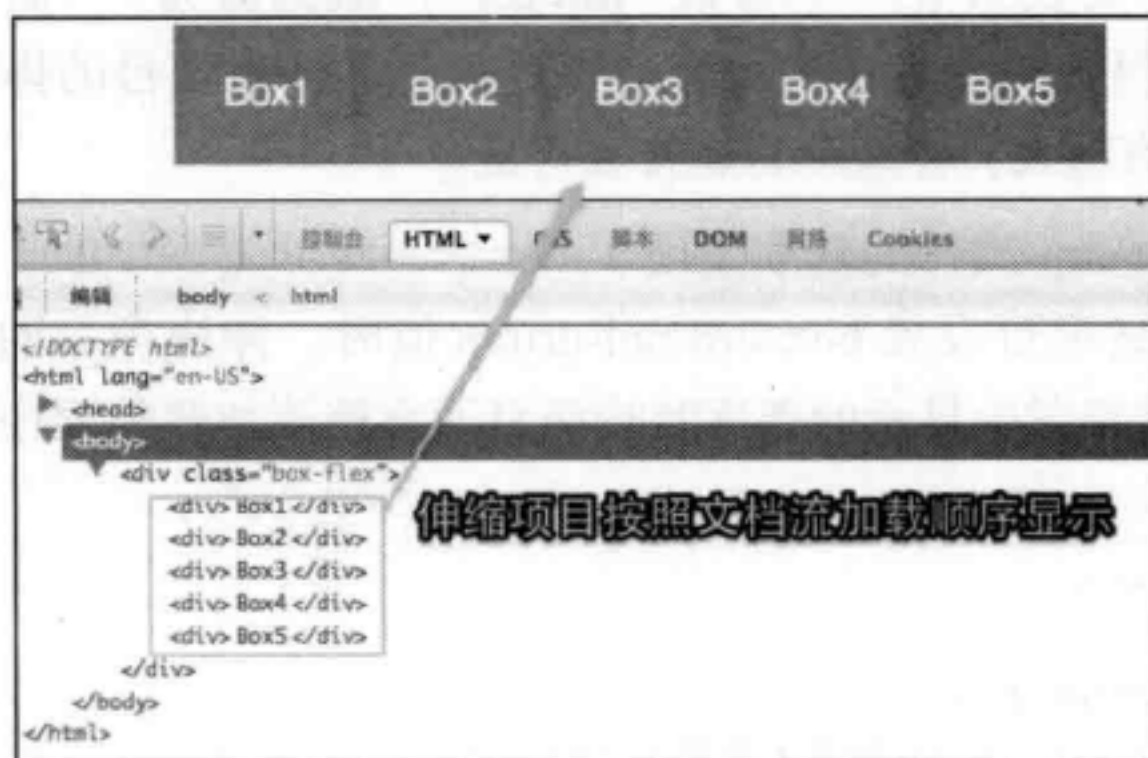


图 8-21 伸缩项目未显式设置 box-ordinal-group 值的效果

接下来为伸缩项目设置 box-ordinal-group 的值，给 box1 和 box4 设置 box-ordinal-group 值为 2，并给 box3 设置 box-ordinal-group 值为 3，如下所示。

```

.box-flex div:nth-child(1),
.box-flex div:nth-child(4){
  background-color: orange;
  -moz-box-ordinal-group: 2;
  -webkit-box-ordinal-group: 2;
  box-ordinal-group: 2;
}
.box-flex div:nth-child(3){
  background-color: green;
  -moz-box-ordinal-group: 3;
  -webkit-box-ordinal-group: 3;
  box-ordinal-group: 3;
}

```

这个时候伸缩容器里的伸缩项目排列顺序就变成 box2、box5、box1、box4、box3。由于 box2 和 box5 的 box-ordinal-group 默认值为 1，而其中 box1 和 box4 显式设置 box-ordinal-group 的值为 2，box5 设置了 box-ordinal-group 的值为 3。伸缩项目的 box-ordinal-group 的值越大，伸缩项目越排在后面。其中 box1 和 box4 的 box-ordinal-group 值同时为 2，不过 box4 在文档流中排在 box1 后面，所以实际显示中 box4 排在 box1 后面。也就是

说当伸缩项目的 `box-ordinal-group` 的值相等时，根据伸缩项目在文档流中的先后顺序排列。其效果如图 8-22 所示。

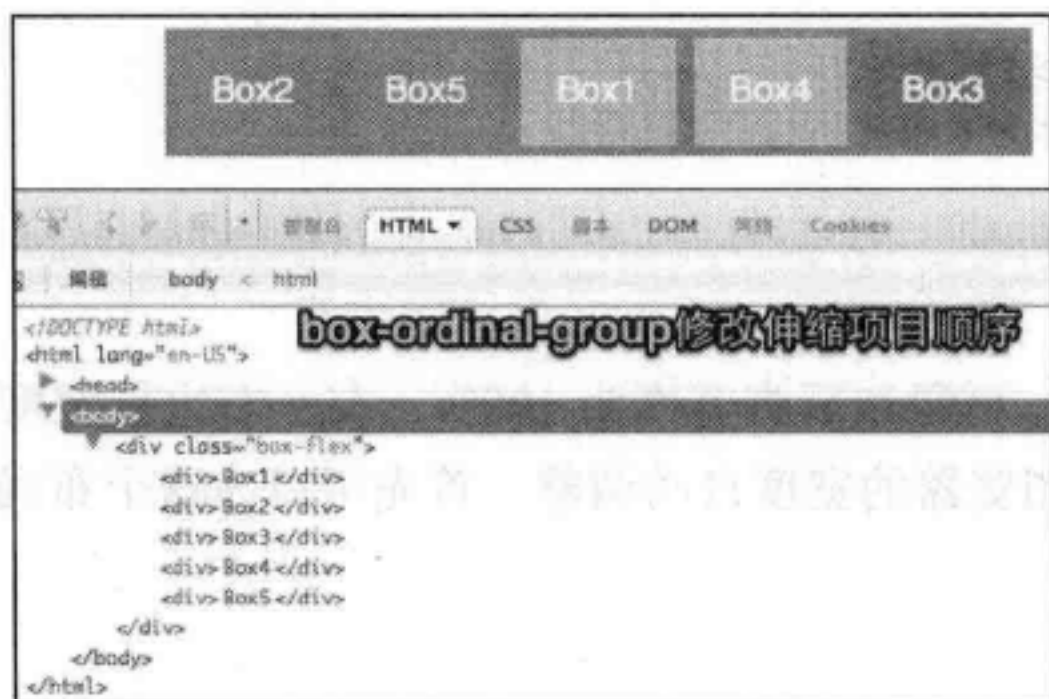


图 8-22 伸缩项目显示设置 `box-ordinal-group` 值，修改伸缩项目的源顺序效果

8.2.9 实战体验：box 制作自适应的三列等高布局

自适应的三列布局是 Web 布局中常见的一种，也常称为流体布局，为了让布局能自适应调整布局中每列的宽度，常使用百分比配合浮动来实现（当然也可使用 `inline-block` 配合百分比实现）。

在使用浮动或者 `inline-block` 配合百分比制作自适应布局的时候，常会碰到一定的难题，例如，需要使用外边距或内边距，计算的时候都比较复杂，但是如果使用 Flexbox 模型 `box` 属性来制作这样的布局，就简单多了。

在制作任何一个布局效果，都离不开 HTML 结构，在这个实例中，使用一个简单的结构模板来制作这个自适应的三列布局，如下所示。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title> 案例实战：box 制作自适应的三列布局 </title>
</head>
<body>
  <div id="header">
    <h1>Header</h1>
  </div>
  <div id="page">
    <div id="main">
      <h1>主内容</h1>
    </div>
    <div id="sidebar-left">
      <h1>左边栏</h1>
    </div>
```

```

    <div id="sidebar-right">
      <h1> 右边栏 </h1>
    </div>
  </div>
  <div id="footer">
    <p> 我是页面页脚 </p>
  </div>
</body>
</html>

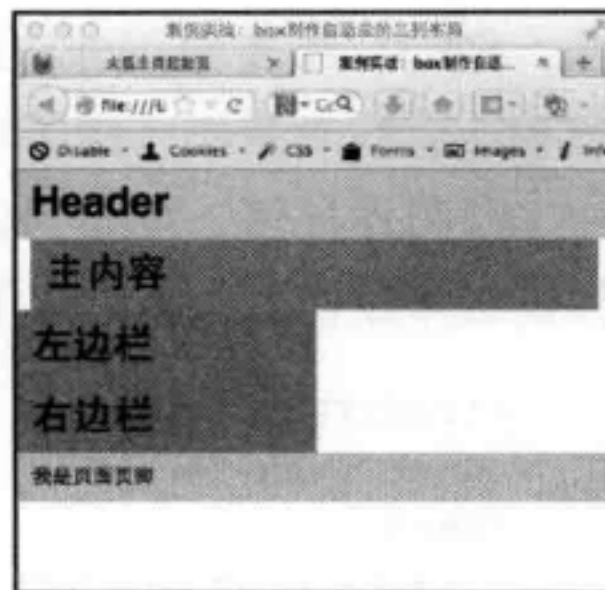
```

假设这个三列布局，页眉和页脚宽度为 100%，左、右边栏宽度为 220px，而主内容为自适应宽度，可以根据浏览器的宽度自动调整。首先可以为这个布局加上一些默认样式。

```

*{
  margin: 0;
  padding: 0;
  -moz-box-sizing: border-box;
  -webkit-box-sizing: border-box;
  box-sizing: border-box;
}
#header,
#footer {
  width: 100%;
  padding: 10px;
  background-color: #ccc;
}
#sidebar-left,
#sidebar-right {
  width: 220px;
  padding: 10px;
  background-color: #f36;
}
#main {
  background-color: #e66;
  padding: 10px;
  margin: 0 10px;
}

```



效果如图 8-23 所示。布局效果自然不是需要的布局效果，图 8-23 三列布局默认效果
左边栏、右边栏都在主内容下面。而需要的左中右三列并排的效果。

为了让布局效果更佳完美，接下来用伸缩布局模型 box 来实现三列自适应布局效果。

```

/*box 制作三列布局*/
#page {
  width: 100%;
  display: -moz-box;
  display: -webkit-box;
  display: box;
}

```



```
#main {
    -moz-box-flex:1;
    -webkit-box-flex:1;
    box-flex:1;
}
```

这时候的布局效果如图 8-24 所示。

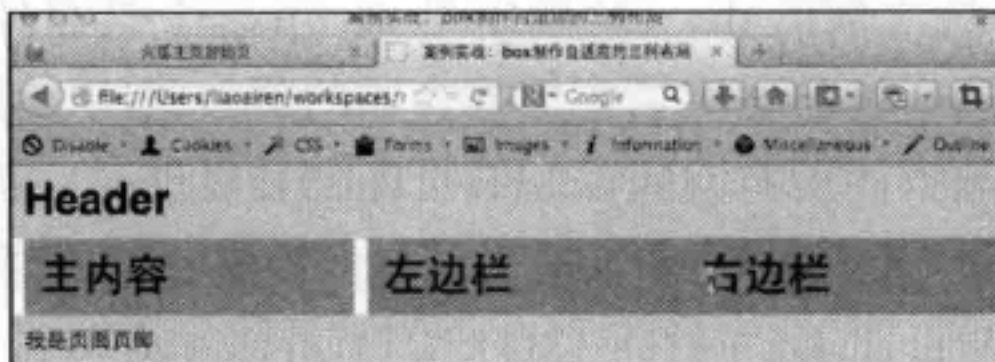


图 8-24 box 实现的三列布局

三列布局效果出来了，只可惜主内容列依旧居左侧，这个时候就需要使用 box 布局模型中的 box-ordinal-group 来重新排列它们的顺序。

```
#main {
    -moz-box-flex:1;
    -webkit-box-flex:1;
    box-flex:1;
    /* 重排主内容顺序 */
    -moz-box-ordinal-group: 2;
    -webkit-box-ordinal-group: 2;
    box-ordinal-group: 2;
}
/* 重排右边栏顺序 */
#sidebar-right{
    -moz-box-ordinal-group: 3;
    -webkit-box-ordinal-group: 3;
    box-ordinal-group: 3;
}
```

这时实现了左边栏在左，主内容居中和右边栏居右的三列布局，居中的主内容列会自动根据浏览器的窗口宽度自适应的调整，如图 8-25 所示。



图 8-25 box 制作的三列自适应布局的效果

图 8-25 的效果说明 box 实现了所需要的三列布局效果，但伸缩布局模型还可以帮助我们实现更完美的三列布局效果。例如在现有的布局基础上实现三列等高并且实现页脚区域

黏附效果。每列的高度都一样，不使用 CSS3 也可以轻松实现^①，而页脚区域黏附指页脚区域固定在浏览器窗口可视区域底部，即使页面的主体内容不足以将页脚区域推到浏览器可视区域的底部，实现这个效果不使用 CSS3 就略有些复杂^②。

不过使用伸缩布局模型来创建页脚区域黏附效果就简单得多了。最关键的技巧是 body 变成一个伸缩容器（在此例是将 body 设置成伸缩容器，别的结构另当别论），并且使用伸缩性属性 box-flex 让页脚区域之前的 div（此例中是 div#page）具有伸缩性。换句话说，页脚前的 div 会根据伸缩容器的高度自适应填充伸缩容器的额外空间，也就是自动拉伸页脚区域的 div 填充浏览器可视区域中的所有空间。如果页面的长度够长，甚至超过了浏览器窗口的可视区域，页脚之前的 div（此例中是 div#page）会保持自己固有的高度，页脚区域也会照常直接显示在其后。

如果希望整个页面的布局要和浏览器窗口可视区域一样高，首先需要保证 html 和 body 元素的高度和浏览器窗口可视区域高度一样高，就算是不包含任何内容也具备这样的效果。

要实现 html 和 body 元素高度和浏览器窗口可视区域高度一样高的效果很简单，只需要显式设置这两个元素的高度为 100%。

```
html,
body {
    height: 100%;
}
```

在这个案例中，body 直接包含了页面布局的页眉 header、页面主体 page 和页脚 footer。接下来，使用 display 属性让包含这三部分的容器 body 变成伸缩容器，也就是将其设置伸缩布局，然后通过 box-orient 属性为这三部分设置为垂直排列。

```
body {
    display: -webkit-box;
    display: -moz-box;
    display: box;
    -moz-box-orient: vertical;
    -webkit-box-orient: vertical;
    box-orient: vertical;
    width: 100%;
}
```

在默认情况之下，header、page 和 footer 三部分按照文档流加载的顺序就是垂直排列的，只不过为了让这三个元素变成伸缩项目，具有伸缩性能适应浏览器可视区域的高度能力。



注意 为了让伸缩容器 body 宽度能与浏览器宽度一样，需要显式设置宽度为 100%。

而在这个布局实例中，希望具有伸缩性的元素就是页面主体元素 page，因此需要在这

① 参见 <http://www.w3cplus.com/css/creaet-height-columns>。

② 参见 CSS2.1 解决方案 <http://www.w3cplus.com/css/css-sticky-foot-at-bottom-of-the-page>。

个元素上设置 box-flex 值大于 1 的正整数,在此例中将其设置为 1。

```
#page{
    -moz-box-flex:1;
    -webkit-box-flex:1;
    box-flex:1;
}
```

现在,按照伸缩布局模型中伸缩原理,扣除页面页眉和页脚区域固有的高度之外,不管页面主体区域 page 内容有多少,这三个区域的高度之和总会至少等于可视区域的高度。因此,页脚区域将会定位在浏览器可视区域的底部,而不紧贴内容区载。另外主体区域 page 本身是另一个伸缩容器,其中的三个元素都自动变成伸缩项目。在伸缩布局模型中,伸缩项目在侧轴的对齐方式 box-align 默认值为 stretch,致使 page 里的三个伸缩项目都会自动拉伸,不管内容高度有多少都具有容器 page 的高度,从而实现三列等高的效果。

至此,几行代码就实现了一个三列适应宽度的布局,而且主内容三列具有等高以及 stick footer 的效果,如图 8-26 所示。

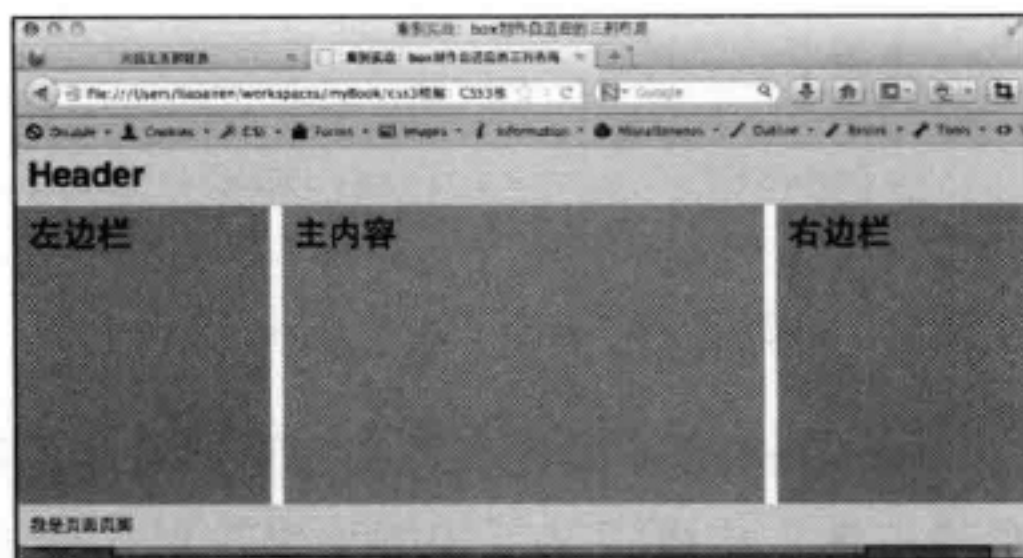


图 8-26 box 制作自适应三列等高布局效果

8.3 混合版本 Flexbox 模型的基本使用

混合版本 Flexbox 模型是伸缩盒模型中的另一个语法规则,主要是对 IE10 浏览器实现伸缩布局的效果,其功能上与最老版本的伸缩布局的盒模型布局大同小异,只是使用细节上略有不同,下面一起来了解 flexbox 的混合语法的使用细节。

8.3.1 伸缩容器设置 display

在混合版本伸缩布局中的伸缩容器设置与旧版本的语法相似,都是使用 display 属性来进行设置,只是所取的值不同。在混合版本伸缩布局时,display 属性所取的值是 flexbox 或者 inline-flexbox。

1. display 属性的语法及参数

display 属性语法如下。

```
display: flexbox | inline-flexbox
```

在 Flexbox 模型中混合版本中伸缩容器的属性设置的参数取值与伸缩布局旧版本中所取的值大同小异,只是所取的关键词不同。在伸缩布局的混合模型布局中主要取值如下。

- ❑ flexbox: 设置元素为块级伸缩容器。
- ❑ inline-flexbox: 设置元素为内联块伸缩容器。

2. display 属性的基本使用

伸缩布局模型混合版本主要是用于 IE 10 浏览器, 在使用混合版本时需要使用浏览器前缀“-ms-”。同样为了把一下元素设置为伸缩容器, 只需要给元素显式设置为 display 属性, 并且将其值设置为 flexbox 或者 inline-flex 值。其中 flexbox 将某元素设置为块伸缩容器, 作用类似于伸缩布局旧版本语法中的 box 属性值; 而取值为 inline-flexbox 时, 将元素设置为行内伸缩容器, 所起的作用类似于伸缩布局旧版本语法中的 inline-box 属性值。

```
.element{ display:-ms-flexbox; /* 设置块级 Flexbox 容器 */ }
```

或者:

```
.element{ display:-ms-inline-flexbox; /* 设置内联级 Flexbox 容器 */ }
```

8.3.2 伸缩流方向 flex-direction

flex-direction 属性和伸缩布局旧版本语法中的 box-orient 属性一样, 主要用来创建伸缩布局主轴, 从而定义了伸缩项目放置在伸缩容器的方向。flex-direction 直接决定了伸缩项目在主轴上的列方式。

1. flex-direction 属性的语法及参数

flex-direction 主要用来决定伸缩布局主轴的方向, 也就是决定伸缩流的方法, 其基本语法如下。

```
-ms-flex-direction: row | row-reverse | column | column-reverse
```

flex-direction 主要通过以下四个属性参数来控制伸缩流方向。

- ❑ row: 此值是 flex-direction 的初始值。伸缩项目按照它们在文档流中出现的顺序进行显示。其中文本书写模式直接影响了文档流的方向, 如果文本的书写模式为 ltr, 伸缩项目在伸缩容器中从左向右排列, 如果文本的书写模式为 rtl, 伸缩项目在伸缩容器中从右向左排列。
- ❑ row-reverse: 此值与 row 值效果相反, 如果书写模式为 ltr, 伸缩项目从右向左排列, 如果书写模式为 rtl, 伸缩项目从左向右排列。
- ❑ column: 伸缩项目按照它们在文档流出现的顺序按列从上到下排列。
- ❑ column-reverse: 与 column 值效果相反, 伸缩项目从下到上排列。

2. flex-direction 属性的基本使用

在 Flexbox 模型布局的旧版本语法规范中, 主要是通过 box-orient 属性来决定伸缩布局的伸缩流方向。不过在 Flexbox 模型布局的混合版本语法范围中, 将由 flex-direction 属性来决定伸缩流方向。接下来通过一些简单的实例来了解 flex-direction 属性的基本使用。

当 flex-direction 取值为 row 时, 类似于 box-orient 中的 horizontal 和 inline-axis 属性, 使伸缩容器中的伸缩项目按照在文档流中出现的先后顺序, 从左向右排列。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>flex-direction 取值为 row 的基本使用 </title>
  <style type="text/css" media="screen">
*{
  margin: 0;
  padding: 0;
}
.flexbox-container {
  padding: 10px;
  border: 1px solid hsla(120,30%,50%,0.8);
  margin: 50px;
  background-color:hsla(10,80%,10%,0.2);
}
.flexbox-container > div {
  width: 100px;
  height: 100px;
  border: 1px solid hsla(120,30%,50%,0.8);
  margin: 5px;
  font-size: 20px;
  text-align: center;
  line-height: 100px;
  color: #fff;
  font-weight:bold;
}
.flexbox-container > div:nth-child(1){
  background-color: hsla(120,30%,50%,0.8);
}
.flexbox-container > div:nth-child(2){
  background-color: hsla(120,30%,10%,0.8);
}
.flexbox-container > div:nth-child(3) {
  background-color: hsla(20,30%,50%,0.8);
}
.flexbox-container > div:nth-child(4){
  background-color:hsla(20,80%,50%,0.8);
}
.flexbox-container > div:nth-child(5) {
  background-color: hsla(320,80%,50%,0.8);
}
.flexbox-container > div:nth-child(6) {
  background-color: hsla(320,80%,80%,0.8);
}
/* 声明伸缩容器 */
.flexbox-container {
```

```

display: -ms-flexbox;
-ms-flex-direction: row;
}
</style>
</head>
<body>
<div class="flexbox-container">
<div>Box1</div>
<div>Box2</div>
<div>Box3</div>
<div>Box4</div>
<div>Box5</div>
<div>Box6</div>
</div>
</body>
</html>

```

其效果如图 8-27 所示。



图 8-27 flex-direction 取值为 row 的效果

当 flex-direction 取值为 column 时，类似于 box-orient 中的 vertical 和 block-axis 效果，使伸缩项目在伸缩容器中按照文档流的先后顺序从上到下排列。

```

/* 声明伸缩容器 */
.flexbox-container {
display: -ms-flexbox;
-ms-flex-direction: column;
}

```

效果如图 8-28 所示。

当 flex-direction 取值为 row-reverse 时，其效果与 box-orient 取值为 horizontal 和 inline-axis，并且 box-direction 取值为 reverse 效果等同。如果文本书写模式为 ltr，伸缩项目在伸缩容器中从右向左排列；如果文本书写模式为 rtl，伸缩项目在伸缩容器中从左向右排列。

```

/* 声明伸缩容器 */
.flexbox-container {
display: -ms-flexbox;
-ms-flex-direction: row-reverse;
}

```

其效果刚好与 flex-direction 取值为 row 时相反，如图 8-29 所示。

当 flex-direction 取值为 column-reverse 时，伸缩项目在伸缩容器中的排列效果与 flex-direction 取值为 column 效果相反。而其效果等同于 box-orient



图 8-28 flex-direction 取值为 column 的效果



图 8-29 flex-direction 取值为 row-reverse 的效果

取值为 vertical 和 block-axis 并且 box-direction 取值为 reverse 时的效果。

```
/* 声明伸缩容器 */
.flexbox-container {
    display: -ms-flexbox;
    -ms-flex-direction: column-reverse;
}
```

其效果如图 8-30 所示。

8.3.3 伸缩换行 flex-wrap

flex-wrap 类似于 box-lines 属性, 使用此属性来启用溢出伸缩容器的伸缩项目自动换行到下一行或自动换列到下一列以及控制伸缩流方向。

1. flex-wrap 属性的语法及参数

flex-wrap 属性的基本语法如下。

```
flex-wrap: nowrap | wrap | wrap-reverse
```

flex-wrap 属性比旧版本 box-lines 属性多一个属性, 除了能控制伸缩项目单行、多行排列之外还有另一个功能, 能让伸缩项目多行反向排列。flex-wrap 各个属性参数简单说明如下。

- ❑ nowrap: 默认值, 类似于 box-lines 属性中的 single。伸缩容器的所有伸缩项目都在一行或一列上显示。如果 display 的 overflow 属性, 可以直接控制伸缩项目在伸缩容器中是否隐藏、裁剪或者出现滚动条。在没有显式地将 flex-wrap 设置为其他属性值时, 一旦伸缩容器没有足够的空间容纳伸缩项目时, 伸缩项目就会溢出伸缩容器。
- ❑ wrap: 伸缩容器的所有伸缩项目在伸缩容器无法容纳时, 伸缩项目在伸缩容器中会自动换行(列), 以多行(列)显示。
- ❑ wrap-reverse: 与 wrap 效果类似, 能让伸缩项目在伸缩容器中无法容纳时, 伸缩项目在伸缩容器中会自动换行(列), 以多行(列)显示。不同的是显示的方向与 wrap 的方向相反。

2. flex-wrap 属性的基本使用

flex-wrap 取值不同时, 其效果不一样, 但其目的是唯一, 主要是用来控制伸缩容器没有足够空间容纳所有伸缩项目时是否单行(列)或多行(列)排列。

当 flex-wrap 不显式设置其他值, 或者显式地设置值为 nowrap 时, 伸缩容器中的所有伸缩项目都在一行或一列上显示。一旦伸缩容器没有足够空间容纳伸缩项目时, 伸缩项目就会溢出伸缩容器。如图 8-31 所示。

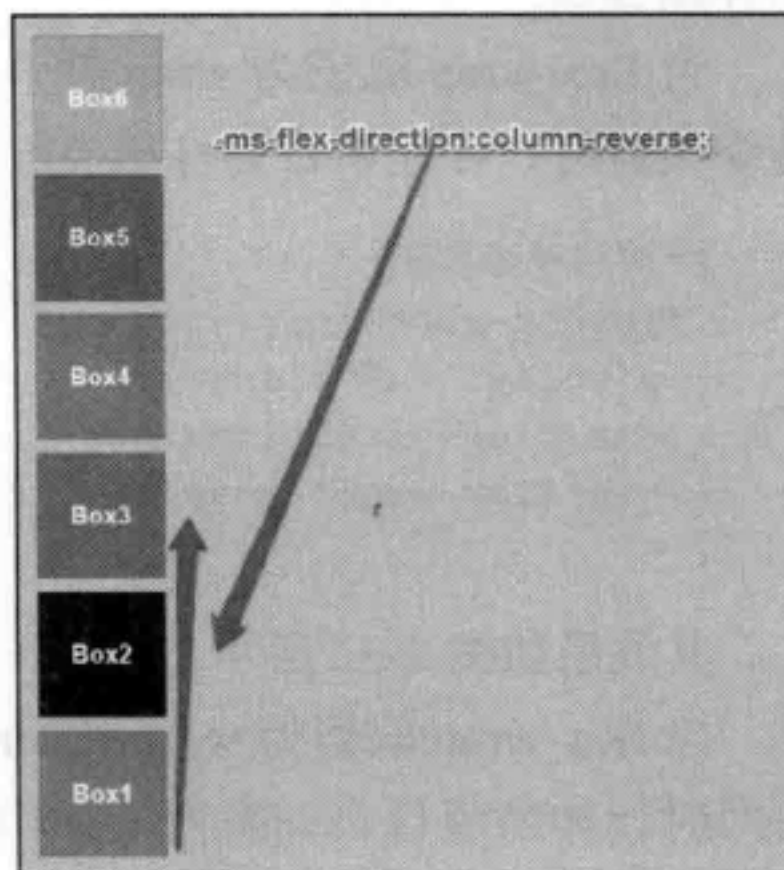


图 8-30 flex-direction 取值为 column-reverse 的效果

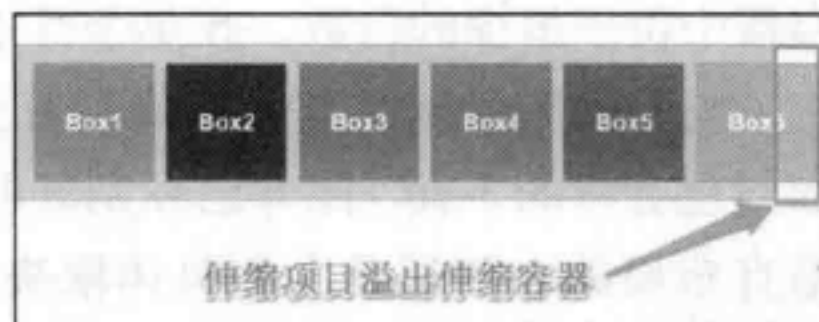


图 8-31 flex-wrap 取值为 nowrap 或未显式设置其他值的效果

如果伸缩项目是按列显示，并且伸缩容器设置了一定的高度时，伸缩项目同样也会溢出伸缩容器。

当 `flex-wrap` 取值为 `wrap` 时，伸缩容器中所有伸缩项目在伸缩容器没有足够空间容纳伸缩项目时，伸缩项目会自动换行。

```
/* 声明伸缩容器 */
.flexbox-container {
  display: -ms-flexbox;
  -ms-flex-direction: row;
  -ms-flex-wrap: wrap;
}
```

其效果如图 8-32 所示。

当 `flex-wrap` 取值为 `wrap-reverse` 值时，同样可以使所有伸缩项目在伸缩容器没有足够空间时让伸缩项目自动换行（列）显示，其效果类似于取值为 `wrap` 的效果，不同的是，伸缩项目在伸缩容器中排列的方向相反。

```
/* 声明伸缩容器 */
.flexbox-container {
  display: -ms-flexbox;
  -ms-flex-direction: row;
  -ms-flex-wrap: wrap-reverse;
}
```

其效果如图 8-33 所示。

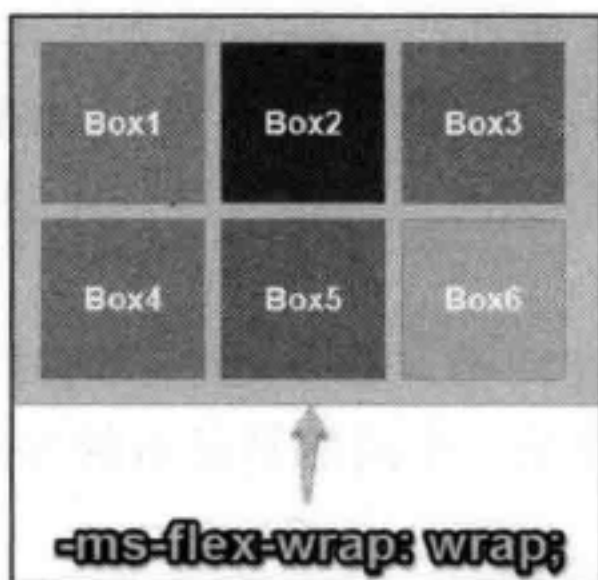


图 8-32 `flex-wrap` 取值为 `wrap` 的效果

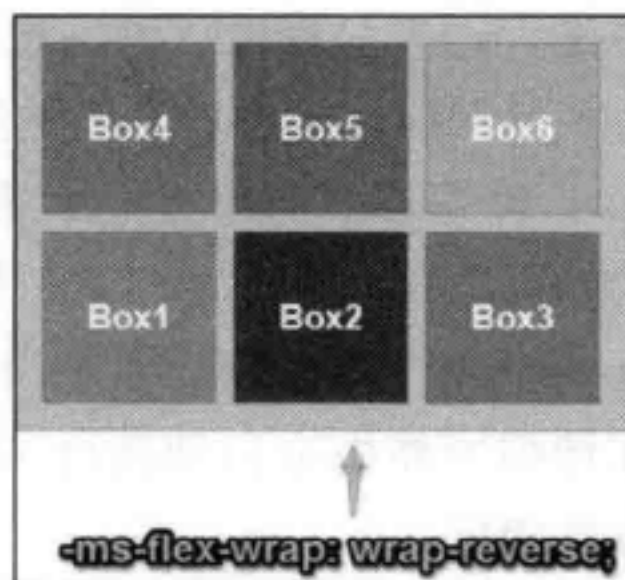


图 8-33 `flex-wrap` 取值为 `wrap-reverse` 的效果

请注意 IE 10 将尝试通过将所有伸缩容器收缩到其最小尺寸，尽可能使伸缩项目在伸缩容器中位于最少的行数。在单个行中无法容纳的单个伸缩项目将在该行的结尾被截断。

按照默认设置，其他行将沿着分块方向添加。从左到右和从右到左布局中，新行将添加到现有行的下面（除非已在别处明确指定了从上到下的分块方向）。同样，新行是出现在垂直布局的右侧还是左侧具体取决于分块方向，而分块方向可能是从左到右或从右到左，具体取决于书写模式或其他特定设置。

8.3.4 伸缩流方向与换行 flex-flow

flex-flow 属性是 Flexbox 模型在混合版本语法规则中新增的一个属性，其主要用来控制伸缩布局中伸缩流方向与伸缩项目的换行。

1. flex-flow 属性的语法及参数

flex-flow 其实是 flex-direction 和 flex-wrap 属性的集合，简单点说是这两个属性的缩写版本，其语法如下所示。

```
flex-flow: <flex-direction> | <flex-wrap>
```

该属性主要包括两个属性值。

- <flex-direction>：用来设置伸缩容器的伸缩流方向。flex-direction 中的所有属性参数都适合用于此处。
- <flex-wrap>：用来设置伸缩容器中的伸缩项目在伸缩容器无足够空间时，伸缩项目在伸缩容器中是否换行排列。flex-wrap 中的所有属性参数都适合用于此处。

2. flex-flow 属性的基本使用

从前面 flex-flow 的语法与属性参数的介绍得知，其实在伸缩盒模型布局中的 flex-flow 属性同时把伸缩流 flex-direction 属性和伸缩项目换行 flex-wrap 属性集于一身。简单说，就是这两个属性的缩写版本，因此 flex-flow 属性时就像 CSS 中其他具有缩写版本的属性，如 border、margin、和 background 等属性一样，在实际使用中不需要在伸缩容器上设置 flex-direction 和 flex-wrap 属性，可以将这两个属性集合体 flex-flow 用于伸缩容器中。根据 flex-flow 中对应的 flex-direction 和 flex-wrap 所取的值，决定伸缩流方向以及伸缩项目在伸缩容器中的排列方式。

```
/* 声明伸缩容器 */
.flexbox-container {
    display: -ms-flexbox;
    -ms-flex-flow: row wrap;
}
```

其实上面示例代码与下面的代码所起的作用是等效的。

```
/* 声明伸缩容器 */
.flexbox-container {
    display: -ms-flexbox;
    -ms-flex-direction: row;
    -ms-flex-wrap: wrap;
}
```

其效果如图 8-32 所示。

8.3.5 主轴对齐 flex-pack

flex-pack 属性和 Flexbox 模型旧的语法规则版本中的 box-pack 属性一样，都是用来设

置伸缩容器当前行伸缩项目在主轴方向的对齐方式，指定如何在伸缩项目之间分布伸缩容器额外空间。

1. flex-pack 属性的语法及参数

flex-pack 的使用语法与 box-pack 语法相同，如下所示。

```
flex-pack: start | end | center | justify | distribute
```

flex-pack 属性除了具有 box-pack 属性的参数之外，在其基础上增添了一个新的属性值 distribute，其中属性参数 start、end、center 和 justify 具体含义，可以参阅前面 box-pack 属性中对应的介绍，此处主要介绍 distribute 属性的含义。

flex-pack 取值为 distribute 时，伸缩项目会平均分布在同一行里，两端保留一半的空间（这里的一半是指中间两相邻伸缩项目之间间距的一半）。如果伸缩容器没有足够空间或者只有一个伸缩项目的时候就会遵循 center 方式。

2. flex-pack 属性的基本使用

flex-pack 和 box-pack 书写语法略有不同而已其在项目中所起的作用与功能是一样的。

当 flex-pack 取值为 start 时，伸缩容器中的第一个伸缩项目的起始边缘置于伸缩容器的主轴起始位置；下一个伸缩项目的起始边缘与第一个伸缩项目的末尾边缘边挨边地放置在一起，其他伸缩项目依此方式沿着布局轴方向（主轴）继续排列。伸缩容器沿着布局轴方向的所有额外空间都被置于布局轴的末尾。

```
/* 声明伸缩容器 */
.flexbox-container {
    display: -ms-flexbox;
    -ms-flex-flow: row wrap;
    -ms-flex-pack: start;
}
```



效果如图 8-34 所示。

图 8-34 flex-pack 取值为 start 的效果

当 flex-pack 取值为 end 时，所起的效果与 start 相反，第一个伸缩项目的末尾边被置于伸缩容器的结束位置；下一个伸缩项目的末尾边缘与第一伸缩项目的起始边缘边挨边的放置在一起；其他的伸缩项目依此方式沿着布局轴方向继续排列。伸缩容器沿着布局轴方向的所有额外空间都被置于布局轴的开始。

```
/* 声明伸缩容器 */
.flexbox-container {
    display: -ms-flexbox;
    -ms-flex-flow: row wrap;
    -ms-flex-pack: end;
}
```

其效果如图 8-35 所示。

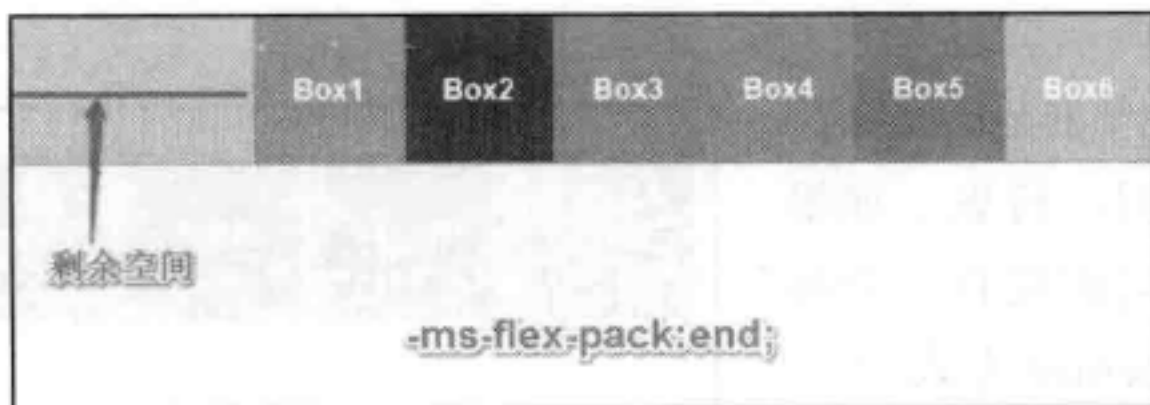


图 8-35 flex-pack 取值为 end 的效果

当 flex-pack 取值为 center 时, 所有伸缩项目边挨边放置在一起, 如同 start 和 end 关键字的描述中所说的那样。但是, 该组伸缩项目在伸缩容器的起始和末尾边缘位于中间位置, 这样伸缩容器所有额外空间都平均分布在第一个伸缩项目的前面和最后一个伸缩项目的后面, 类似水平居中, 也常用这种方法制作水平居中效果。

```
/* 声明伸缩容器 */
.flexbox-container {
    display: -ms-flexbox;
    -ms-flex-flow: row wrap;
    -ms-flex-pack: center;
}
```

其效果如图 8-36 所示。



图 8-36 flex-pack 取值为 center 的效果

当 flex-pack 取值为 justify 时, 伸缩项目会平均地分布在行里。第一个伸缩项目一行中的开始位置, 最后一个伸缩项目在一行终点位置。也就是第一个伸缩项目的起始边缘位于伸缩容器的开始位置; 最后一个伸缩项目的末尾边缘与伸缩容器的结束位置边挨边地放置在一起, 所有其他伸缩项目都位于第一个和最后一个伸缩项目之间。这样, 沿着伸缩容器布局轴方向的任何额外空间都被平均分布于各个伸缩项目之间。

```
/* 声明伸缩容器 */
.flexbox-container {
    display: -ms-flexbox;
    -ms-flex-flow: row wrap;
    -ms-flex-pack: justify;
}
```

其效果如图 8-37 所示。

当 flex-pack 取值为 distribute 时，伸缩项目平均分布在同一行里。如果伸缩容器没有足够空间或只有一个伸缩项目时，将会遵循 center 方式。

```
/* 声明伸缩容器 */
.flexbox-container {
  display: -ms-flexbox;
  -ms-flex-flow: row wrap;
  -ms-flex-pack: distribute;
}
```

其效果如图 8-38 所示。

8.3.6 侧轴对齐 flex-align

混合版本规范中的 flex-align 属性和 Flexbox 旧版本规范中的 box-align 所起的作用相同，主要用来定义伸缩项目可以在伸缩容器的当前行的侧轴上对齐方式。可以把它想象成侧轴（垂直于主轴）的 flex-pack 属性。

1. flex-align 属性的语法及参数

flex-align 属性和 box-align 属性所起的作用相同，用来决定伸缩项目在侧轴的对齐，其语法如下所示。

```
flex-align: start | end | center | baseline | stretch
```

flex-align 属性除了具有 box-align 属性的参数之外，还增添了另外一个属性值 stretch。其中 flex-align 属性值 start、end、center 和 baseline 详细含义可以参数 box-align 的相关属性，此处只介绍 flex-align 新增添的 stretch 属性值。

当 flex-align 属性取值为 stretch 时，伸缩项目拉伸填充整个伸缩容器，如果伸缩容器没有足够的空间，将以 start 方式显示。

2. flex-align 属性的基本使用

当 flex-align 取值为 start 时，伸缩项目在侧轴起点边的外边距紧靠住该行在侧轴起始的边。如果伸缩容器具有 row 或 column 的 -ms-flex-direction 计算值，每个伸缩项目的起始边缘（或基线）与伸缩容器的起始边缘对齐。布局轴垂直的任何额外空间（伸缩容器在侧轴的额外空间）都放置在每个伸缩项目的末尾边缘之后。如果伸缩容器具有 row-reverse 或 column-reverse 的 -ms-flex-direction 计算值，每个伸缩项目的末尾边缘（或基线）与伸缩容器的末尾边缘对齐。布局轴垂直的任何额外空间（伸缩容器在侧轴的额外空间）都将放置在每个伸缩项目的起始边缘之前。

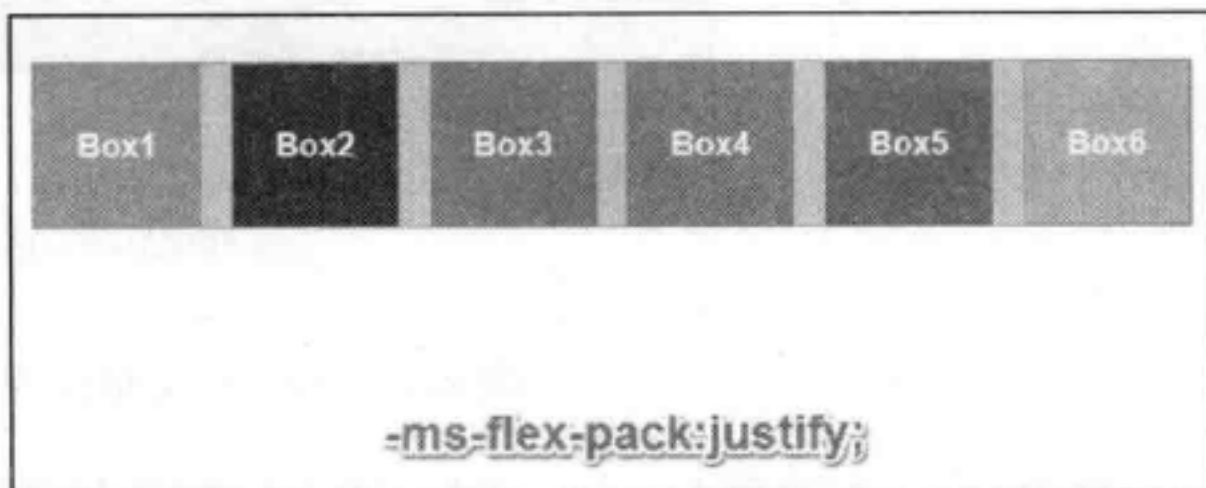


图 8-37 flex-pack 取值为 justify 的效果

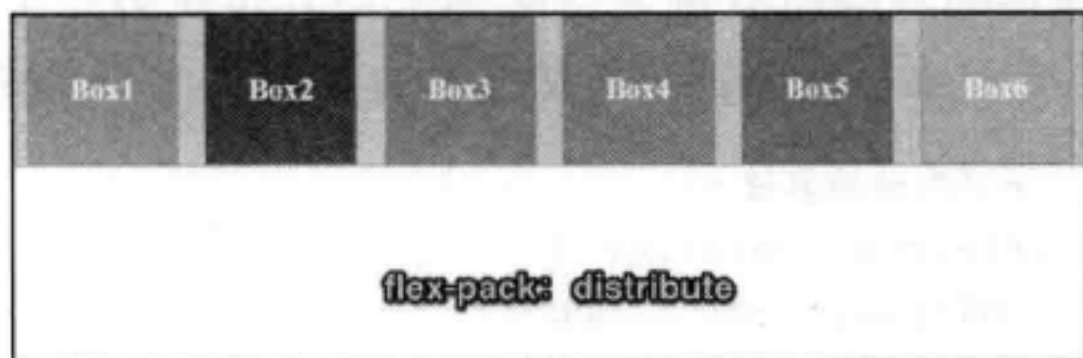


图 8-38 flex-pack 取值为 distribute 的效果


```

<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>flex-align 取值为 start 的基本使用 </title>
  <style type="text/css" media="screen">
*{
  margin: 0;
  padding: 0;
}
.flexbox-container {
  padding: 10px;
  border: 1px solid hsla(120,30%,50%,0.8);
  margin: 50px;
  background-color:hsla(10,80%,10%,0.2);
}
.flexbox-container > div {
  width: 100px;
  height: 100px;
  border: 1px solid hsla(120,30%,50%,0.8);
  margin: 5px;
  font-size: 20px;
  text-align: center;
  line-height: 100px;
  color: #fff;
  font-weight:bold;
}
.flexbox-container > div:nth-child(1){
  background-color: hsla(120,30%,50%,0.8);
}
.flexbox-container > div:nth-child(2){
  background-color: hsla(120,30%,10%,0.8);
}
.flexbox-container > div:nth-child(3) {
  background-color: hsla(20,30%,50%,0.8);
}
.flexbox-container > div:nth-child(4){
  background-color:hsla(20,80%,50%,0.8);
}
.flexbox-container > div:nth-child(5) {
  background-color: hsla(320,80%,50%,0.8);
}
.flexbox-container > div:nth-child(6) {
  background-color: hsla(320,80%,80%,0.8);
}
/* 声明伸缩容器 */
.flexbox-container {
  display: -ms-flexbox;
  -ms-flex-flow: row wrap;
  -ms-flex-align:start;

```

```

}
</style>
</head>
<body>
  <div class="flexbox-container">
    <div>Box1</div>
    <div>Box2</div>
    <div>Box3</div>
    <div>Box4</div>
    <div>Box5</div>
    <div>Box6</div>
  </div>
</body>
</html>

```

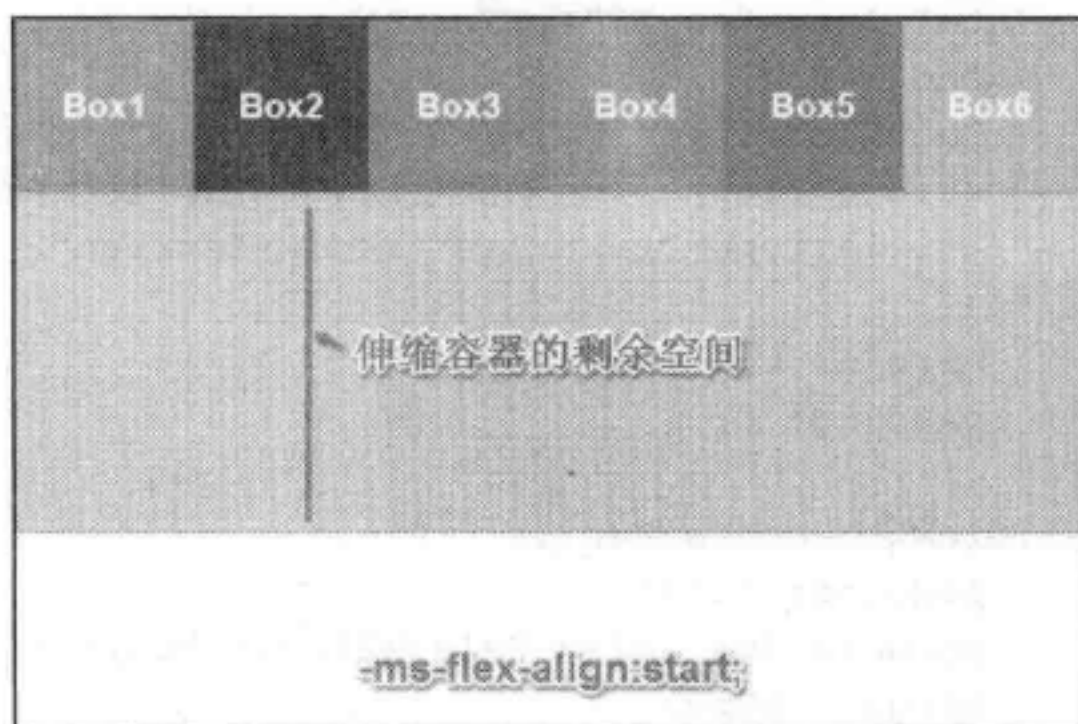


图 8-39 flex-align 取值为 start 的效果

其效果如图 8-39 所示。

当 flex-align 取值为 end 时，伸缩项目在侧轴终点边的外边距靠住该行在侧轴终点的边。如果伸缩容器有 row 或 column 的 -ms-flex-direction 计算值，每个伸缩项目的末尾边缘与伸缩容器的末尾边缘对齐。布局轴垂直的额外空间（伸缩容器侧轴额外空间）都放置在每个伸缩项目的起始边缘之前。如果伸缩容器具有 row-reverse 或 column-reverse 的 -ms-flex-direction 计算值，每个伸缩项目的起始边缘与伸缩项目的起始边缘对齐。布局轴垂直的任何额外空间（伸缩容器侧轴的额外空间）都放置在每个伸缩项目末尾边缘之后。

```

/* 声明伸缩容器 */
.flexbox-container {
  display: -ms-flexbox;
  -ms-flex-flow: row wrap;
  -ms-flex-align: end;
}

```

其效果如图 8-40 所示。

当 flex-align 取值为 center 时，伸缩项目的外边距盒在该行的侧轴上居中放置。每个伸缩项目都居中在伸缩容器的起始边缘和末尾边缘之间。布局轴垂直的额外空间都均匀分布在每个伸缩项目之前和之后。

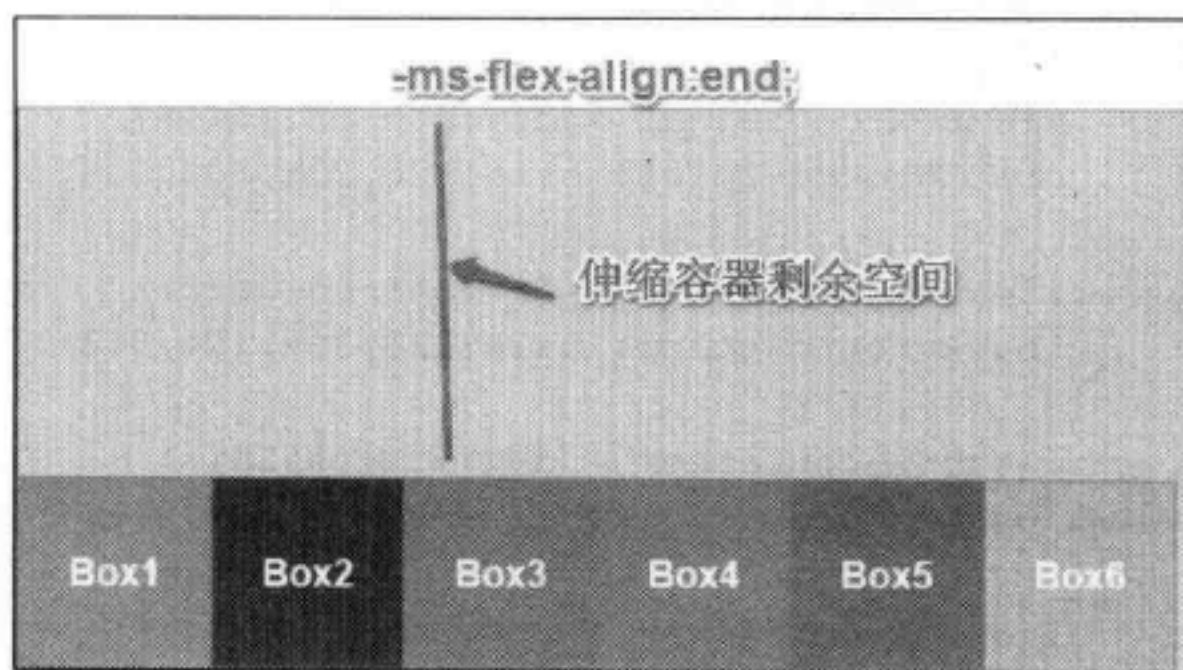


图 8-40 flex-align 取值为 end 的效果

```

/* 声明伸缩容器 */
.flexbox-container {
  display: -ms-flexbox;
  -ms-flex-flow: row wrap;
  -ms-flex-align: center;
}

```

其效果如图 8-41 所示。

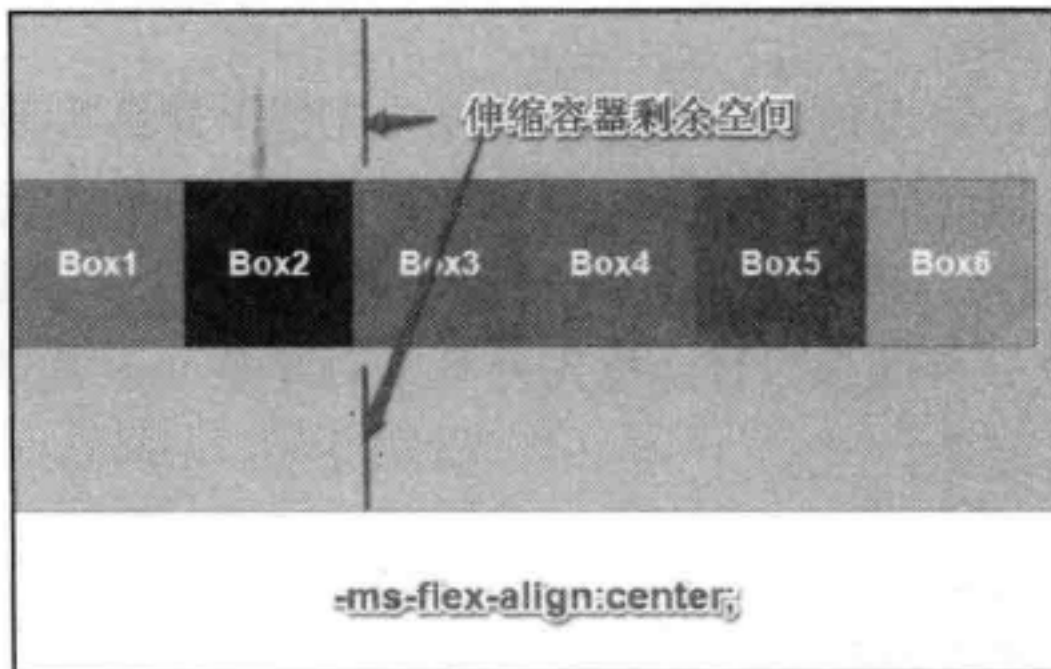


图 8-41 flex-align 取值为 center 的效果

当 flex-align 取值为 baseline 时，伸缩项目根据它们的基线对齐。所有伸缩项目的基线（取决于 -ms-flex-direction 属性的起始边缘和末尾边缘）都彼此对齐。占用空间最多且垂直于布局轴的子元素遵循 start 规则，然后所有剩余伸缩项目的基线与该伸缩项目的基线对齐。

```
.flexbox-container > div:nth-child(1){
  background-color: hsla(120,30%,50%,0.8);
  line-height:50px;
}
.flexbox-container > div:nth-child(2){
  background-color: hsla(120,30%,10%,0.8);
  line-height: 100px;
}
.flexbox-container > div:nth-child(3) {
  background-color: hsla(20,30%,50%,0.8);
  line-height: 150px;
}
.flexbox-container > div:nth-child(4){
  background-color:hsla(20,80%,50%,0.8);
  line-height: 200px
}
.flexbox-container > div:nth-child(5) {
  background-color: hsla(320,80%,50%,0.8);
  line-height: 150px;
}
.flexbox-container > div:nth-child(6) {
  background-color: hsla(320,80%,80%,0.8);
  line-height: 100px;
}
/* 声明伸缩容器 */
.flexbox-container {
  display: -ms-flexbox;
  -ms-flex-flow: row wrap;
  -ms-flex-align:baseline;
}
```


其效果如图 8-42 所示。

当 `flex-align` 取值为 `stretch` 时，伸缩项目拉伸填充整个伸缩容器。此值会使项目的外边距盒的尺寸在遵照 (`min/max-width/height`) 属性的限制下尽可能接近所在行的尺寸。将对每个伸缩项目进行拉伸以便完全填充垂直于布局轴的可用空间。如果设置了伸缩项目的 `max-width` 或 `max-height` 属性，伸缩项目将会优先并且布局遵循 `start` 规则。

```
/* 声明伸缩容器 */
.flexbox-container {
    display: -ms-flexbox;
    -ms-flex-flow: row wrap;
    -ms-flex-align: stretch;
}
```

其效果如图 8-43 所示。

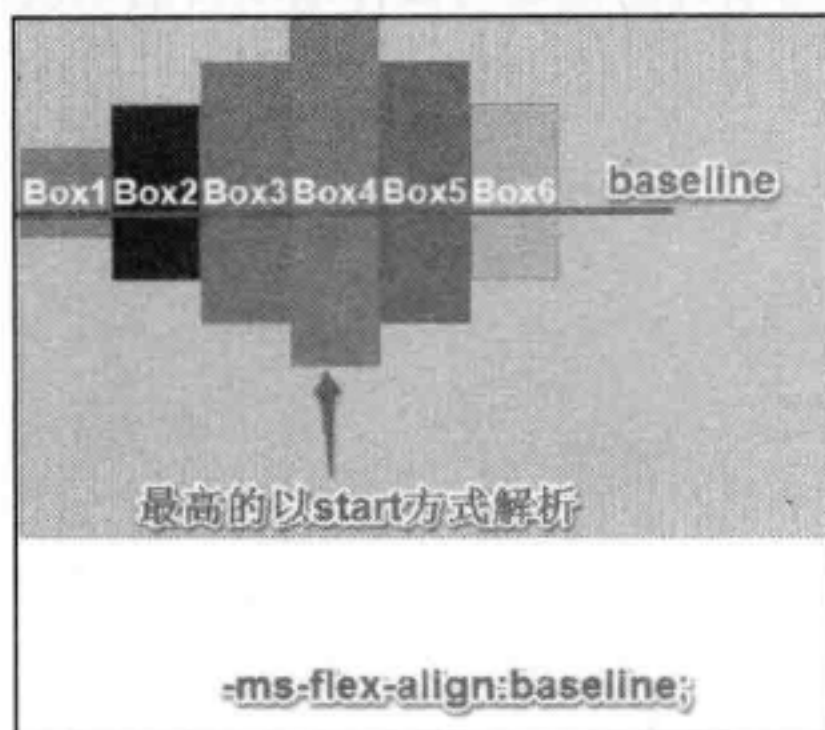


图 8-42 `flex-align` 取值为 `baseline` 的效果

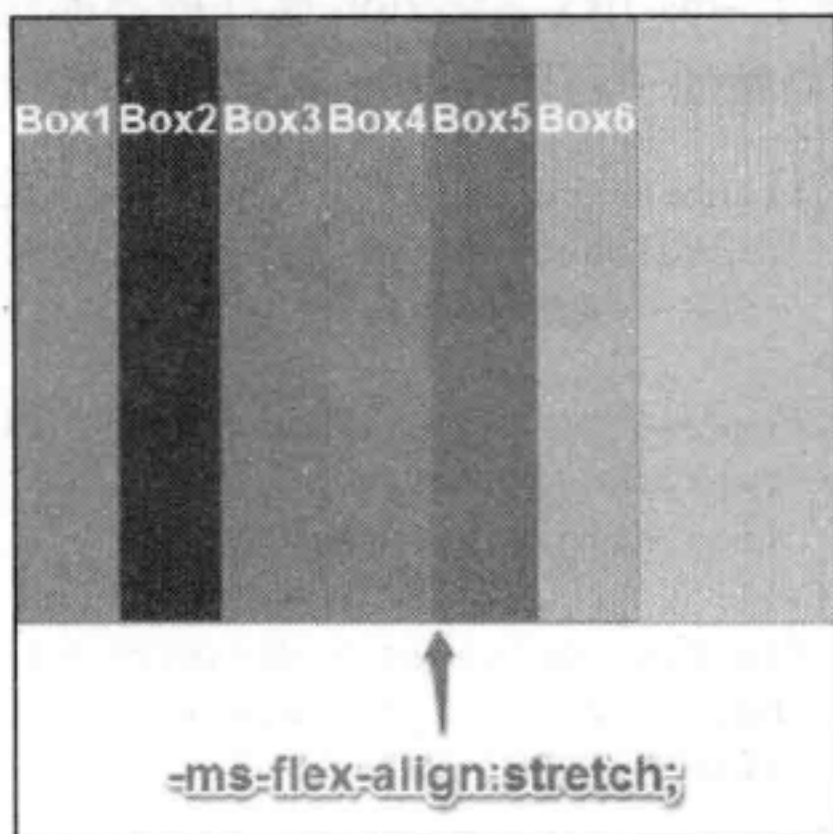


图 8-43 `flex-align` 取值为 `stretch` 的效果

请注意 `flex-align` 属性值都是依赖于文本的排版格式，伸缩容器和伸缩项目的起始边缘和末尾边缘都取决于布局方向。例如，对于从左到右的布局来说，起始边缘是伸缩容器的上边缘；对于从上到下布局来说，起始边缘是伸缩容器的左边缘等。同样，对于从左到右布局来说，伸缩项目的末尾边缘是下边缘；对于从上到下布局来说，伸缩项目的末尾边缘是右边缘等。

8.3.7 堆栈伸缩行 `flex-line-pack`

在 Flexbox 模型中，`flex-pack` 属性设置了伸缩项目在伸缩布局主轴的排列方式；`flex-align` 属性设置了伸缩项目在伸缩布局侧轴的方式。这两个属性都是针对于单行伸缩项目在伸缩容器中的排列方式，但往往有时候伸缩项目一行无法完全显示，需要多行（列）排列显示。此时伸缩布局在前面的属性基础上新增了另一个属性 `flex-line-pack`。

`flex-line-pack` 属性主要用来定义伸缩容器中伸缩项目行在侧轴的对齐方式，类似于 `flex-pack` 属性，只不过这个属性是用来控制伸缩项目行在布局轴的堆放方式。

1. `flex-line-pack` 属性的语法及参数

`flex-line-pack` 属性的使用 and `flex-pack` 一样，其语法如下。

```
flex-line-pack: start | end | center | justify | distribute | stretch
```

`flex-line-pack` 属性参数与 `flex-pack` 属性参数相比，增添了 `stretch` 属性。其各属性参数值含义如下。

- `start`：伸缩项目行向布局轴（伸缩容器的侧轴）的起始位置靠齐。
- `end`：伸缩项目行向布局轴（伸缩容器的侧轴）的结束位置靠齐。
- `center`：伸缩项目行向布局轴（伸缩容器的侧轴）的中间位置靠齐。
- `justify`：伸缩项目行平均分布在由局轴（伸缩容器的侧轴），第一个伸缩项目行在侧轴的顶部，最后一个伸缩项目行在侧轴的底部。
- `distribute`：伸缩项目行平均分布在布局轴（伸缩容器的侧轴），两端保留一半的空间；
- `stretch`：伸缩项目行拉伸填充整个伸缩容器。

2. `flex-line-pack` 属性的基本使用

`flex-line-pack` 属性主要用来控制伸缩项目行在侧轴的排列方式，它和 `flex-pack` 略有不同。`flex-pack` 针对的是个体，而 `flex-line-pack` 针对的是群体。

`flex-line-pack` 取值为 `start` 时，伸缩项目行向布局轴的起始位置靠齐。第一行伸缩项目的起始边缘置于伸缩容器的布局轴起始点位置，下一行伸缩项目的起始边缘与第一行伸缩项目的末尾边缘边挨边地放置在一起，其他的伸缩项目行依此方式沿着布局轴方向继续排列。伸缩容器沿着布局轴的所有额外空间都置于布局轴的末尾。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>flex-line-pack 取值为 start 的效果</title>
  <style type="text/css" media="screen">
    *{
      margin: 0;
      padding: 0;
    }
    .flexbox {
      margin: 0.25em 5em;
      border: 1px solid hsla(120, 30%, 50%, .8);
      background-color: hsla(10, 80%, 10%, .2);
      height: 250px;
    }
    .flexbox:nth-child(2n){
      margin-bottom: 200px;
```

```

    }
    .flexbox > div {
        border: 1px solid hsla(120,30%,50%,.8);
        font: bold 2em arial;
        color: #fff;
        width: 150px;
        padding: .2em;
    }
    .flexbox > div:nth-child(odd){
        background-color: hsla(120,30%,50%,.8);
    }
    .flexbox > div:nth-child(even){
        background-color: hsla(120,30%,10%,.8);
    }
    .flexbox > div:last-child{
        background-color: hsla(20,10%,60%,.8);
    }
    /* 设置伸缩容器 */
    .flexbox {
        display: -ms-flexbox;
        -ms-flex-wrap: wrap;
    }
    .flex-direction-column {
        -ms-flex-direction: column;
    }
    .flex-line-pack-start {
        -ms-flex-line-pack: start;
    }
</style>
</head>
<body>
    <div class="flexbox flex-line-pack-start">
        <div>box1</div>
        <div>box2</div>
        <div>box3</div>
        <div>box4</div>
        <div>box5</div>
        <div>box6</div>
        <div>box7</div>
        <div>box8</div>
        <div>Start</div>
    </div>
    <div class="flexbox flex-line-pack-start flex-direction-column">
        <div>box1</div>
        <div>box2</div>
        <div>box3</div>
        <div>box4</div>

```



```

    <div>box5</div>
    <div>box6</div>
    <div>box7</div>
    <div>box8</div>
    <div>Start</div>
  </div>

</body>
</html>

```

其效果如图 8-44 所示。

`flex-line-pack` 取值为 `end` 时，伸缩项目行向布局轴的结束位置靠齐，刚好与取值为 `start` 相反。第一个伸缩项目行的末尾边被置于伸缩容器的结束位置；下一个伸缩项目行的末尾边缘与第一个伸缩项目行的起始边缘边挨边的放置在一起；其他伸缩项目行依此方式沿着布局轴方向继续排列。伸缩容器沿着布局轴方向所有额外空间都被置于布局轴的开始。

```

/*flex line-pack*/
.flex-line-pack-end {
  -ms-flex-line-pack:end;
}

```

其效果如图 8-45 所示。

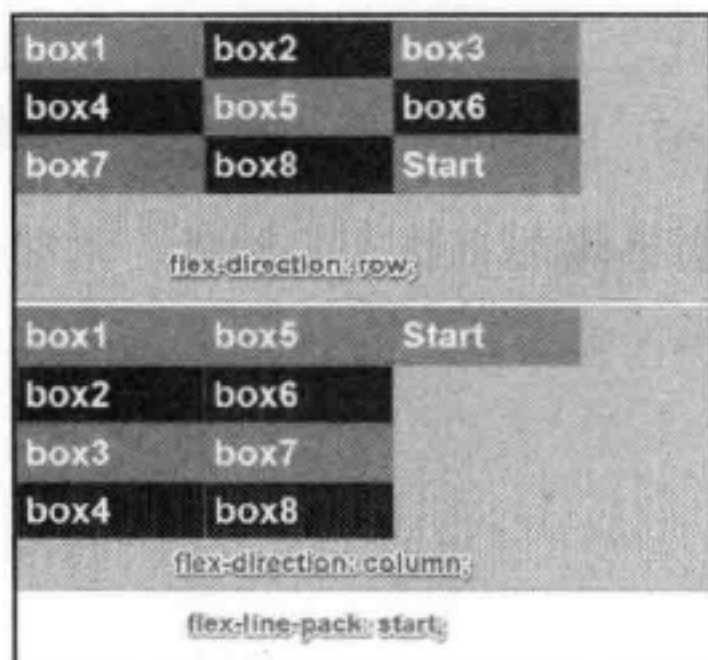


图 8-44 `flex-line-pack` 取值为 `start` 的效果

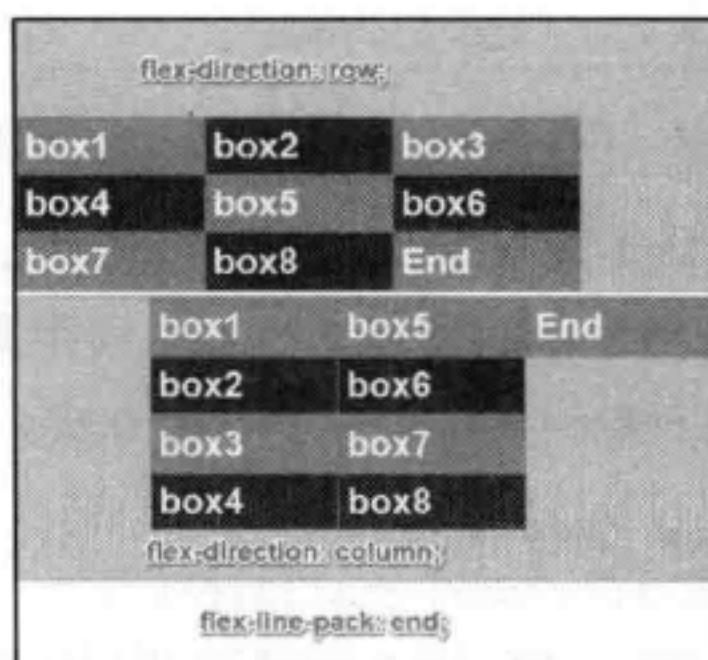


图 8-45 `flex-line-pack` 取值为 `end` 的效果

`flex-line-pack` 取值为 `center` 时，伸缩项目行向布局轴的中间位置靠齐。所有伸缩项目行边挨边放置在一起，如同 `start` 和 `end` 关键词的描述中所说的那样。但是，该组伸缩项目行在伸缩容器的起始和末尾边缘之间位于中间位置，这样伸缩容器所有额外空间都平均分布在第一行伸缩项目前面和最后一行伸缩项目的后面。

```

/*flex line-pack*/
.flex-line-pack-center {
  -ms-flex-line-pack:center;
}

```

其效果如图 8-46 所示。

`flex-line-pack` 取值为 `justify` 时，伸缩项目行会平均分布在布局轴里。第一伸缩项目行始终在伸缩容器的侧轴的开始处，最后伸缩项目行始终在伸缩容器的侧轴的结束位置。沿着布局轴方向的任何额外空间都被平均分布在各伸缩项目行之间。如果伸缩容器没有足够的空间，或者只有一行伸缩项目时，将会遵循 `start` 方式。

```
/*flex line-pack*/
.flex-line-pack-justify {
  -ms-flex-line-pack:justify;
}
```

其效果如图 8-47 所示。

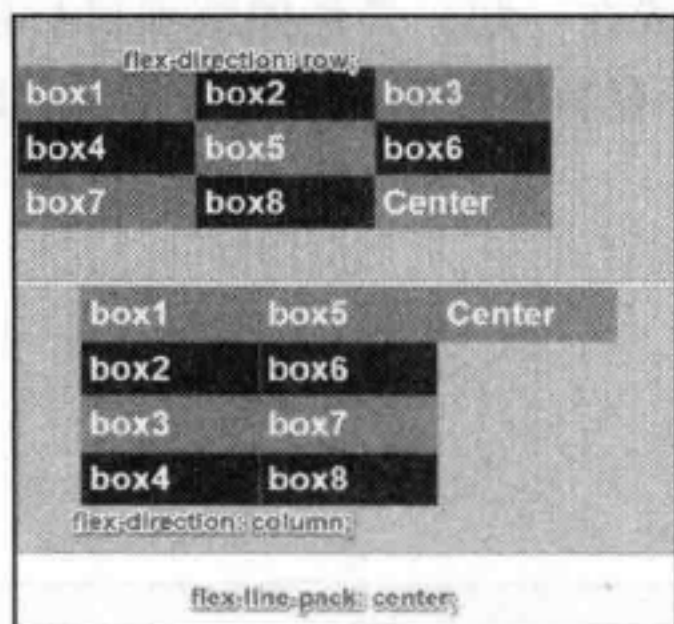


图 8-46 `flex-line-pack` 取值为 `center` 的效果

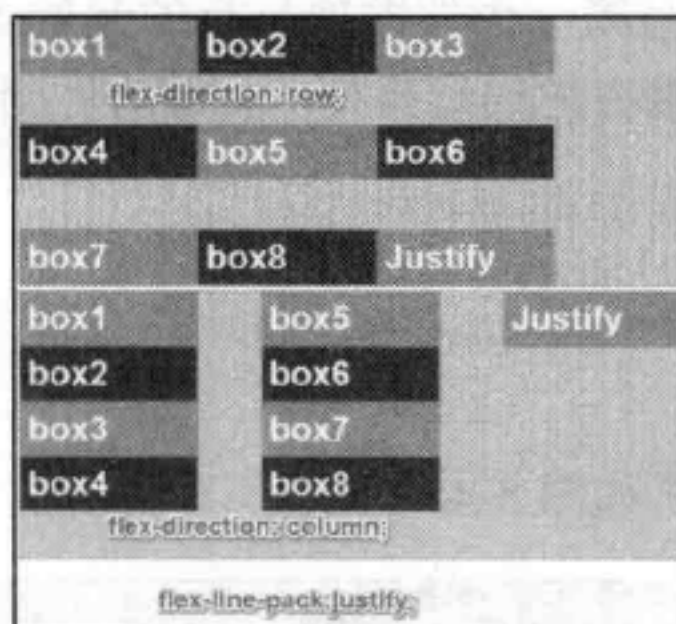


图 8-47 `flex-line-pack` 取值为 `justify` 的效果

`flex-line-pack` 取值为 `distribute` 时，伸缩项目行会平均分布在布局轴里，两端保留一半的空间且等于中间伸缩项目行中间距离的一半。如果伸缩容器没有足够空间或者只有一个伸缩项目行的时候会遵循 `center` 方式。

```
/*flex line-pack*/
.flex-line-pack-distribute {
  -ms-flex-line-pack:distribute;
}
```

其效果如图 8-48 所示。

`flex-line-pack` 取值为 `stretch` 时，伸缩项目行会位伸填充整个伸缩容器。如果伸缩容器没有足够的空间，将遵循 `start` 方式。

```
/*flex line-pack*/
.flex-line-pack-stretch {
  -ms-flex-line-pack:stretch;
}
```

其效果如图 8-49 所示。

这样的效果。来看看伸缩项目扩大和收缩适合伸缩容器的可用空间如下所示。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>-ms-flex 的基本使用 </title>
  <style type="text/css" media="screen">
*{
  margin: 0;
  padding: 0;
}
.flexbox-container {
  border: 1px solid hsla(120,30%,50%,0.8);
  margin: 50px;
  background-color:hsla(10,80%,10%,0.2);
  height: 300px;
}
.flexbox-container > div {
  border: 1px solid hsla(120,30%,50%,0.8);
  font-size: 20px;
  text-align: center;
  color: #fff;
  font-weight:bold;
}
.flexbox-container > div:nth-child(1){
  background-color: hsla(120,30%,50%,0.8);
}
.flexbox-container > div:nth-child(2){
  background-color: hsla(120,30%,10%,0.8);
}
.flexbox-container > div:nth-child(3) {
  background-color: hsla(20,30%,50%,0.8);
}
.flexbox-container > div:nth-child(4){
  background-color:hsla(20,80%,50%,0.8);
}
/* 声明伸缩容器 */
.flexbox-container {
  display: -ms-flexbox;
  -ms-flex-direction:row;
  -ms-flex-wrap: wrap;
  -ms-flex-pack:center;
  -ms-flex-align:center;
}
.flexbox-container > div {
  -ms-flex:1;
}
.flexbox-container > div:hover {
  -ms-flex:2;
}
```

```

</style>
</head>
<body>
  <div class="flexbox-container">
    <div>Box1</div>
    <div>Box2</div>
    <div>Box3</div>
    <div>Box4</div>
  </div>
</body>
</html>

```

在这个实例中，没有显式地设置 box1 ~ box4 四个元素根据浏览器宽度自动调整大小。使用 flex 属性让 box1 ~ box4 四个 div 元素占有 1flex 单元。因为没有给这四个 div 元素明确的设置宽度，而每个 div 元素都有相同的宽度。修改浏览器的窗口大小，div 元素将会扩展或收缩。

在这个例子中，已经设置了一样的高度，但是使用同样的方法也可以实现伸缩性。可能不会总是希望所有元素是相同的大小，如在悬浮状态下，设置元素为 2flex 单元。

```

.flexbox-container > div:hover {
  -ms-flex:2;
}

```

现在可用的空间除以 5，而在悬浮状态是占有 2 份。注意，一个元素的 2flex 单元并不一定就是 1flex 单元宽度的两倍。它只获得了添加 2 倍比例到其可用空间的首先宽度。在示例中，首先宽度是 0（默认状态下），其最后效果如图 8-50 所示。

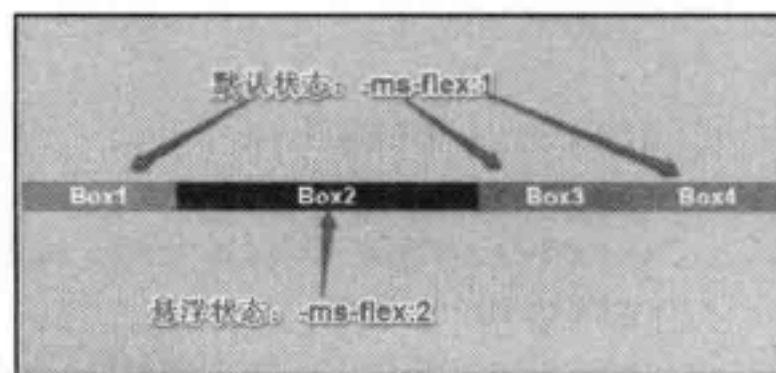


图 8-50 -ms-flex 的使用效果

8.3.9 显示顺序 flex-order

默认情况下，伸缩项目是按照文档流出先后顺序排列。然而，flex-order 属性和 box-ordinal-group 属性一样可以控制伸缩项目在它们的伸缩容器出现的顺序。值必须是大于 0 的整数。

1. flex-order 属性的语法及参数

flex-order 属性的语法很简单。

```
flex-order: <integer>
```

<integer> 是一个自然数，默认值为 0，用来设置伸缩项目的位置顺序号。伸缩项目的值越大，越排在后面。

2. flex-order 属性的基本使用

-ms-flex-order 属性使你能将伸缩项目放置在序数组中，从序数组 0 开始。每个序数组

中的所有伸缩项目都沿着指定的布局轴呈现在一个序数组中的任何伸缩项目之前。因此，序数组 0 中的所有伸缩项目都呈现在序数组 1 的任何伸缩项目之前，依此类推。序数组以内的伸缩项目按照 DOM 顺序呈现。按照默认设置，其他行将沿着分块的方向添加。

例如，以下标记指定四个伸缩项目并使用 `-ms-flex-order` 属性为它们中的大多数分配序数值。

- ❑ Box1 的 `-ms-flex-order` 值为 1。
- ❑ Box2 的 `-ms-flex-order` 值为 0，这是初始值。
- ❑ Box3 的 `-ms-flex-order` 值为 0。
- ❑ Box4 没有显式设置 `-ms-flex-order` 值，这意味着它被设置为 `-ms-flex-order` 的初始值为 0。

来先看一个简单的实例。

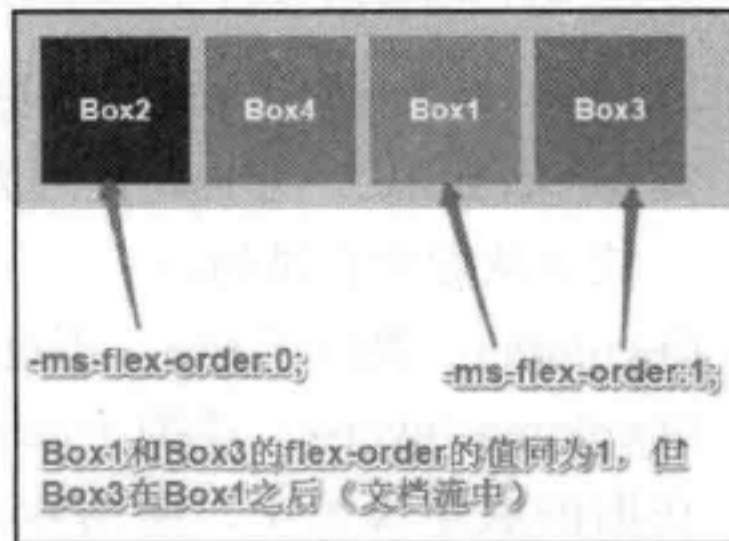
```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>-ms-flex-order 的基本使用 </title>
  <style type="text/css" media="screen">
*{
  margin: 0;
  padding: 0;
}
.flexbox-container {
  padding: 10px;
  border: 1px solid hsla(120,30%,50%,0.8);
  margin: 50px;
  background-color:hsla(10,80%,10%,0.2);
}
.flexbox-container > div {
  width: 100px;
  height: 100px;
  border: 1px solid hsla(120,30%,50%,0.8);
  margin: 5px;
  font-size: 20px;
  text-align: center;
  line-height: 100px;
  color: #fff;
  font-weight:bold;
}
.flexbox-container > div:nth-child(1){
  background-color: hsla(120,30%,50%,0.8);
  -ms-flex-order: 1;
}
.flexbox-container > div:nth-child(2){
  background-color: hsla(120,30%,10%,0.8);
  -ms-flex-order: 0;
```



```

}
.flexbox-container > div:nth-child(3) {
  background-color: hsla(20,30%,50%,0.8);
  -ms-flex-order: 1;
}
.flexbox-container > div:nth-child(4){
  background-color:hsla(20,80%,50%,0.8);
}
/* 声明伸缩容器 */
.flexbox-container {
  display: -ms-flexbox;
  -ms-flex-direction: row;
}
</style>
</head>
<body>
  <div class="flexbox-container">
    <div>Box1</div>
    <div>Box2</div>
    <div>Box3</div>
    <div>Box4</div>
  </div>
</body>
</html>

```



效果如图 8-51 所示。

图 8-51 flex-order 更改伸缩项目顺序

Box2 和 Box4 都位于序数组 0 中, 而 Box1 和 Box3 都位于序数组 1 中。因为在每个序数组内, 伸缩项目都按照 DOM 顺序呈现, 所以这些伸缩项目在伸缩容器中按以下顺序显示: Box2、Box4、Box1 和 Box3。

8.4 新版本 Flexbox 模型的基本使用

Flexbox 是一个新的盒子模型, 主要优化了 UI 布局。经过几年的发展, Flexbox 具有多个版本, Firefox 和 Safari 在使用最旧版本的语法, IE 规范更改了语法。而在 2012 年 9 月, W3C 为 Flexbox 推出了其最新版本语法。

较前两个 Flexbox 规范版相比, Flexbox 伸缩布局功能和各种术语概念并无差异, 只是部分属性的语法略有不同, 接下来一起来看看新版本 Flexbox 模型的基本使用。

8.4.1 伸缩容器 display

Flexbox 最新规范中设置元素为伸缩容器的方法和前两个版本规范设置方法是一样的, 都是通过 display 属性设置, 只不过所取的值不同而已。

```
display: flex | inline-flex
```

其中属性值 `flex` 等同于 `box` 和 `flexbox`，将一个容器设置为块伸缩容器，而属性值 `inline-flex` 等同于 `inline-box` 和 `inline-flexbox`，将一个容器设置为内联伸缩容器。



注意 CSS 的 `columns` 在伸缩容器上没有效果；`float`、`clear` 和 `vertical-align` 在伸缩项目上没有效果。

8.4.2 伸缩流方向 `flex-direction`

`flex-direction` 适用于伸缩容器，也就是伸缩项目的父元素。主要用来创建主轴，从而定义了伸缩项目放置在伸缩容器的方向。

```
flex-direction: row | row-reverse | column | column-reverse
```

- ❑ `row`：在 `ltr` 排版方式下从左向右排列；在 `rtl` 排版方式下从右向左排列（默认值）。
- ❑ `row-reverse`：与 `row` 排列方向相反，在 `ltr` 排版方式下从右向左排列；在 `rtl` 排版方式下从左向右排列。
- ❑ `column`：类似于 `row`，不过是从上到下排列。
- ❑ `column-reverse`：类似于 `row-reverse`，不过是从下到上排列。

在旧的版本规范中，使用 `box-direction` 属性设置为 `reverse` 和在新的规范版本中设置 `row-reverse` 或 `column-reverse` 得到的效果相同。如果想要的效果是 `row` 或 `column` 可以省略不设置，因为 `normal` 是默认的初始值。

设置 `direction` 为 `reverse`，主轴就会翻转。意思是，当使用 `ltr` 书写模式的，指定 `row-reverse` 时，所有伸缩项目会从右向左排列。类似地，`column-reverse` 将会使所有伸缩项目从下向上排列，来代替从上往下排列。

在旧的规范版本中，需要使用 `box-orient` 来设置书写模式的方向。当使用 `ltr` 模式时，`horizontal` 可用在 `inline-axis`，`vertical` 可用 `block-axis`。如果使用的是一个自上而下的书写模式，这些值就会翻转。

8.4.3 伸缩换行 `flex-wrap`

`flex-wrap` 属性适用于伸缩容器，也就是伸缩项目的父元素。主要用来定义伸缩容器里是单行还是多行显示，侧轴的方向决定了新行堆放的方向。

```
flex-wrap: nowrap | wrap | wrap-reverse
```

- ❑ `nowrap`：伸缩容器单行显示，`ltr` 排版下，伸缩项目从左到右排列；`rtl` 排版上伸缩项目从右向左排列（默认值）。
- ❑ `wrap`：伸缩容器多行显示，`ltr` 排版下，伸缩项目从左到右排列；`rtl` 排版上伸缩项目从右向左排列。

- ❑ `wrap-reverse`：伸缩容器多行显示，ltr 排版下，伸缩项目从右向左排列；rtl 排版下，伸缩项目从左到右排列（和 `wrap` 相反）。

在旧的规范版本中，使用 `box-lines` 来设置伸缩容器的伸缩项目是单行或多行显示。`wrap-reverse` 让伸缩项目在侧轴上进行 `start` 和 `end` 翻转，所以，如果伸缩项目水平排列，伸缩项目翻转不会到新的一行下面，它会翻转到一个新的行上。（简单理解就是伸缩项目只能上下或前后翻转顺序。）

8.4.4 伸缩流方向与换行 `flex-flow`

`flex-flow` 适用于伸缩容器，也就是伸缩项目的父元素。这是 `flex-direction` 和 `flex-wrap` 属性的缩写版本，同时定义了伸缩容器的主轴和侧轴。其默认值为 `row nowrap`。

```
flex-flow: <'flex-direction'> || <'flex-wrap'>
```

写本书的时候，在 Firefox 中并不支持 `flex-wrap` 或者 `box-lines` 属性。

显式设置伸缩项目在主轴上的对齐方式，可以使用 `flex-flow` 属性。默认值为 `row`，伸缩项目沿着主轴水平排列。如果想让伸缩项目沿着侧轴（垂直于主轴）排列，可以将 `flex-flow` 设置为 `column`。当 `row` 和 `column` 属性值添加后缀 “`-reverse`” 将会反向排列（`flex-flow: row-reverse` 或者 `flex-flow: column-reverse`）。

另一个大家要注意，`flex-flow` 属性和 `writing-mode` 有直接的关系。当使用 `writing-mode: vertical-rl` 时转向垂直布局（如传统的中文、日文和韩文排版，也就是竖排），`flex-flow: row` 将垂直排列伸缩项目，和 `column` 将水平排列伸缩项目。

8.4.5 主轴对齐 `justify-content`

主轴对齐 `justify-content` 属性适用于伸缩容器，也就是伸缩项目的父元素。主要用来定义伸缩项目沿着主轴线的对齐方式。当一行的所有伸缩项目都不能伸缩或可伸缩但已经达到其最大长度时，这一属性才会对伸缩容器额外空间进行分配。当伸缩项目溢出某一行时，这一属性也会在项目的对齐上施加一些控制。

```
justify-content: flex-start | flex-end | center | space-between | space-around
```

- ❑ `flex-start`：伸缩项目向一行的起始位置靠齐。类似旧的规范版本中的 `start`（默认值）。
- ❑ `flex-end`：伸缩项目向一行的结束位置靠齐。类似旧的规范版本中的 `end`。
- ❑ `center`：伸缩项目向一行的中间位置靠齐。类似旧的规范版本中的 `center`。
- ❑ `space-between`：伸缩项目会平均地分布在行里。第一个伸缩项目一行中的最开始位置，最后一个伸缩项目在一行中最终点位置。类似旧的规范版本中的 `justify`。
- ❑ `space-around`：伸缩项目会平均地分布在行里，两端保留一半的空间。类似旧的规范版本中的 `distribute`。

flex-start、flex-end 和 center 一看就懂。space-between 和 space-around 则是分配伸缩项目之间空白空间的不同方法，如图 8-52 所示。

在 IE 10 中，这个属性称做 flex-pack，在旧的浏览器中称为 box-pack（再一次需要使用浏览器的私有前缀）。justify-content 属性具有 flex-start、flex-end、center、space-between 和 space-around 五个属性值。在 IE 10 和旧的规范中，它们分别是 start、end、center、justify 和 distribute（在旧的规范中不支持 distribute）。flex-start 表示左对齐（如果书写模式是 rtl 表示右对齐），flex-end 表示右对齐（如果书写模式是 ltr 表示左对齐），space-between 表示伸缩项目平均分布在布局轴上，space-around 均匀分布在布局轴上，而且第一个伸缩项目前和最后一个伸缩项目后的空间为中间伸缩项目间距的一半。

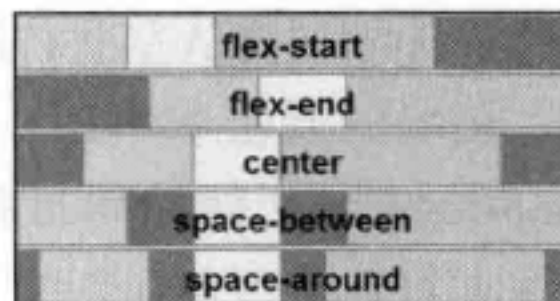


图 8-52 展示 justify-content 属性各关键词对伸缩项目的效果

8.4.6 侧轴对齐 align-items 和 align-self

Flexbox 模型中，设置伸缩项目在侧轴的对齐方式有两种。一种是伸缩项目行在侧轴的对齐方式，主要由属性 align-items 控制；另一种是伸缩项目自身在侧轴的对齐方式，主要由属性 align-self 控制。

align-items 和 justify-content 相呼应。align-items 调整伸缩项目在侧轴上的定位方式。主要用来定义伸缩项目可以在伸缩容器的当前行的侧轴上对齐方式。可以把它想象成侧轴（垂直于主轴）的 justify-content。

align-items: flex-start | flex-end | center | baseline | stretch

- ❑ flex-start：伸缩项目在侧轴起点边的外边距紧靠住该行在侧轴起始边。
- ❑ flex-end：伸缩项目在侧轴终点边的外边距靠住该行在侧轴终点边。
- ❑ center：伸缩项目的外边距盒在该行的侧轴上居中放置。
- ❑ baseline：伸缩项目根据伸缩项目的基线对齐。
- ❑ stretch：默认值，伸缩项目拉伸填充整个伸缩容器。此值会使用伸缩项目的外边距盒的尺寸在遵照（min/max-width/height）属性的限制下尽可能接近所在行的尺寸。

和之前一样，flex-start、flex-end 和 center 的意义显而易见。stretch 也很简单，会将伸缩项目从侧轴起点拉伸到侧轴终点。baseline 则是让伸缩项目与它们的基线对齐，基线根据伸缩项目的内容计算得到。W3C 标准的图 8-53 很好地解释了这些属性值的意义。

在 IE 10 中，这个属性称为 flex-align，在旧的语法规范中称为 box-align。align-items 属性的属性值包括 flex-start、flex-end、center、baseline 和 stretch，在旧的规范版本中对应

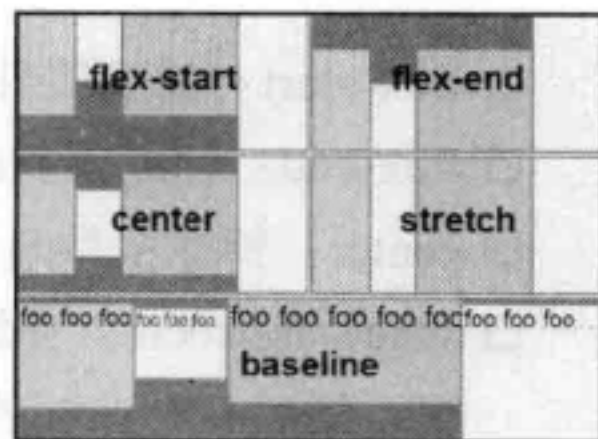


图 8-53 展示 align-items 属性各关键词对伸缩项目的效果

的属性值为 start、end、center、baseline 和 stretch。

伸缩项目的 align-self 属性主要是用来设置单独伸缩项目在侧轴的对齐方式，可以用来覆盖该伸缩项目的伸缩容器的 align-items 属性。它的值和 align-items 一样。

对于匿名伸缩项目，align-self 的值永远与其关联的伸缩容器的 align-items 的值相同。另外，如果伸缩项目的任一个侧轴上的外边距为 auto，则 align-self 没有效果。如果 align-self 的值为 auto，则其计算值为伸缩项目的伸缩容器的 align-items 值，如果该伸缩项目没有伸缩容器，则计算值为 stretch。

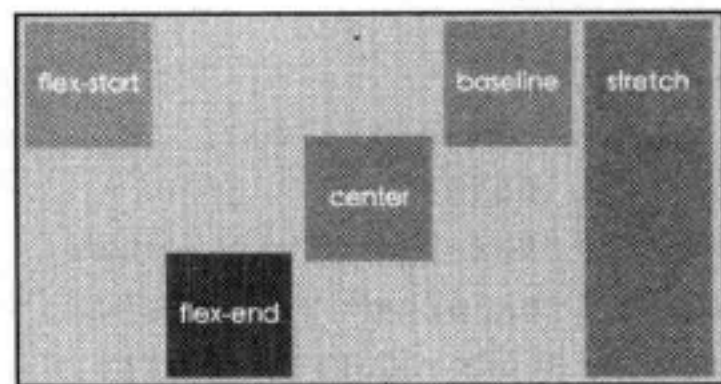
接下来看一个简单的实例，展示伸缩项目 align-self 取不同值的效果。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>align-self 的基本使用 </title>
<style type="text/css" media="screen">
*{
  margin: 0;
  padding: 0;
}
.flex-container {
  border: 1px solid hsla(120,30%,50%,0.8);
  margin: 50px;
  background-color:hsla(10,80%,10%,0.2);
  height: 300px;
  width: 565px;
  /* 声明伸缩容器 */
  display: -webkit-flex;
  display: flex;
  -webkit-flex-flow: row wrap;
  flex-flow: row wrap;
  -webkit-align-items: start;
  align-items: start;
}
.flex-container > div {
  width: 100px;
  min-height: 100px;
  border: 1px solid hsla(120,30%,50%,0.8);
  margin: 5px;
  font-size: 20px;
  text-align: center;
  line-height: 100px;
  color: #fff;
  font-weight: bold;
}
.flex-container > div:nth-child(1){
  background-color: hsla(120,30%,50%,0.8);
  -webkit-align-self: flex-start;
```

```

    align-self: flex-start;
}
.flex-container > div:nth-child(2){
    background-color: hsla(120,30%,10%,0.8);
    -webkit-align-self: flex-end;
    align-self: flex-end;
}
.flex-container > div:nth-child(3) {
    background-color: hsla(20,30%,50%,0.8);
    -webkit-align-self: center;
    align-self: center;
}
.flex-container > div:nth-child(4){
    background-color:hsla(20,80%,50%,0.8);
    -webkit-align-self: baseline;
    align-self: baseline;
}
.flex-container > div:nth-child(5) {
    background-color: hsla(320,80%,50%,0.8);
    -webkit-align-self: stretch;
    align-self: stretch;
}
</style>
</head>
<body>
    <div class="flex-container">
        <div>flex-start</div>
        <div>flex-end</div>
        <div>center</div>
        <div>baseline</div>
        <div>stretch</div>
    </div>
</body>
</html>

```



其效果如图 8-54 所示。

图 8-54 展示 align-self 属性关键词对伸缩项目的效果

8.4.7 堆栈伸缩行 align-content

align-content 属性会更改 flex-wrap 的行为。它和 align-items 相似，但不是对齐伸缩项目，它对齐的是伸缩行。当伸缩容器的侧轴还有额外空间时，align-content 属性可以用来调准伸缩行在伸缩容器里的对齐方式，这与调准伸缩项目在主轴上对齐方式的 justify-content 属性类似。简单点说，align-content 属性主要用来调准伸缩行在伸缩容器里的对齐方式。可能已经想到了，它接受的值也很相似。

```

align-content: flex-start | flex-end | center | space-between | space-around |
stretch

```


- ❑ **flex-start**: 各行向伸缩容器的起点位置堆叠。
- ❑ **flex-end**: 各行向伸缩容器的结束位置堆叠。
- ❑ **center**: 各行向伸缩容器的中间位置堆叠。
- ❑ **space-between**: 各行在伸缩容器中平均分布。
- ❑ **space-around**: 各行在伸缩容器中平均分布, 在两边各有一半的空间。
- ❑ **stretch**: align-content 的默认值, 各行将会伸展以占用额外的空间。

在 IE 10 中, 这个属性称做 flex-line-pack。不过在旧版本规范中没有这一属性。align-content 属性主要包括 flex-start、flex-end、center、space-between、space-around 和 stretch。在 flex-line-pack 属性中, 与 align-content 对应的值为 start、end、center、justify、distribute 和 stretch。下面来自 W3C 标准的图 8-55 很好地解释了这些属性值的意义。

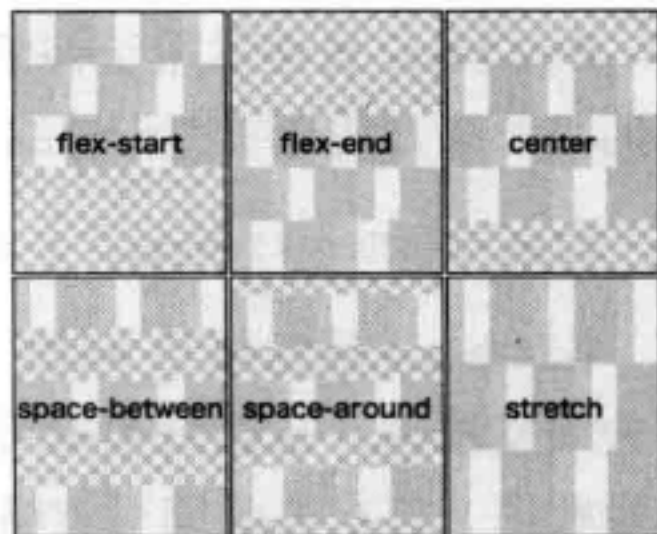


图 8-55 展示 align-content 各关键词对多行伸缩项目的效果



提示 这个属性只有伸缩项目有多行时才生效, 这种情况只有 display 的 flex-wrap 为 wrap 时, 并且没有足够的空间把伸缩项目行放在同一行中。也就是这个属性将对每一行起作用而不是每个伸缩项目。

8.4.8 伸缩性 flex

伸缩布局决定性的特性是让伸缩项目可伸缩, 也就是让伸缩项目的宽度或高度自动填充伸缩容器额外的空间, 这可以用 flex 属性完成。一个伸缩容器会等比地按照各伸缩项目的扩展比率分配额外空间, 也会按照收缩比率缩小各伸缩项目以避免伸缩项目溢出伸缩容器。

```
flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]
```

flex 属性可以用来指定可伸缩长度的部件: 扩展比率、收缩比率, 以及伸缩基准值。当一个元素是伸缩项目时, flex 属性将代替主轴长度属性决定元素的主轴长度。若元素不是伸缩项目, 则 flex 属性没有效果。

如果所有伸缩项目的 flex-grow 设置为 1, 每个伸缩项目将设置为一个大小相等的额外空间。如果给其中一个伸缩项目设置 flex-grow 值为 2, 这个伸缩项目所占的额外空间是其他伸缩项目所占额外空间的 2 倍。

```
flex-grow: <number> (默认值为: 0)
```

flex-shrink 属性适用于伸缩项目, 此属性根据需要用来定义伸缩项目收缩的能力。

```
flex-shrink: <number> (默认值为: 1)
```

`<number>` 部件可以用来设置 `flex-shrink` 长度并指定收缩比率，也就是额外空间是负值的时候，此收缩项目相对于伸缩容器里其他伸缩项目能收缩的空间比例。若省略此部件，它会被设置为 1。在收缩的时候收缩比率会以伸缩基准值加权。



注意 `flex-shrink` 的负值同样生效。

`flex-basis` 属性适用于伸缩项目。这个属性值用来设置伸缩项目的伸缩基准值，伸缩容器的额外空间按比率进行伸缩。

`flex-basis: <length> | auto` (默认值为: `auto`)

`flex-basis` 属性值与 `width` 属性使用相同的值的此部件可以用来设置 `flex-basis` 长度并指定伸缩基准值，也就是根据可伸缩比率计算出额外空间的分布之前，伸缩项目主轴长度的起始数值。若在 `flex` 缩写省略此部件，`flex-basis` 的指定值是长度是 0。若 `flex-basis` 的指定值是 `auto`，则伸缩基准值的指定值是元素主轴长度属性的值。

来看一个来自于 W3C 标准规范的示例截图，显示绝对伸缩（以零为基准值开始）与相对伸缩（以项目的内容大小为基准值开始），这三个伸缩项目的伸缩比例分别是 1、1、2，如图 8-56 所示。

`flex` 取值为 `none` 时，相当于 `flex-grow` 值为 0，`flex-shrink` 值为 0 和 `flex-basis` 值为 `auto`，也就是 `flex: none` 时相当于 `flex: 0 0 auto`。

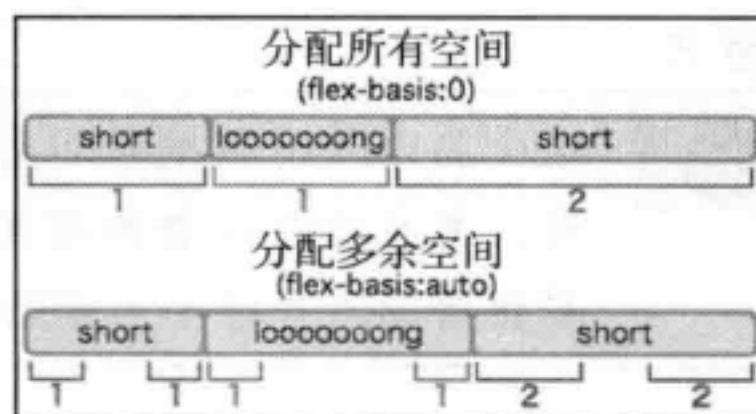


图 8-56 `flex` 分配伸缩空间示意图

上面介绍了 `flex` 属性取值情况，接下来看 `flex` 属性常见值的效果。

- ❑ `flex: 0 auto` 和 `flex: initial` 与 `flex: 0 1 auto` 相同。这也就是 `flex` 的初始值。根据 `width/height` 属性决定元素的尺寸。如果项目的主轴长度属性的计算值为 `auto`，则会根据其内容来决定元素尺寸。当伸缩容器额外的空间为正值时，伸缩项目无法伸缩，但当额外空间不足时，伸缩项目可收缩至其最小值。可以通过用伸缩项目的对齐相关的属性以及 `margin` 属性的 `auto` 值控制伸缩项目沿着主轴的对齐方式。
- ❑ `flex: auto` 与 `flex: 1 1 auto` 相同。根据 `width/height` 属性决定元素的尺寸，但是完全可以伸缩，会吸收主轴上额外的空间。如果所有伸缩项目均为 `flex: auto`、`flex: initial` 或 `flex: none`，则在项目尺寸决定后，额外的正空间会被平分给是 `flex: auto` 的项目。
- ❑ `flex: none` 与 `flex: 0 0 auto` 相同。根据 `width/height` 属性决定元素的尺寸，但是完全不可伸缩。其效果与 `initial` 类似，但即使在空间不够而溢出的情况下，伸缩项目也不能收缩。
- ❑ `flex: <positive-number>` 与 `flex: <positive-number> 10px` 相同。该值使元素可伸缩，并将伸缩基准值设置为 0，导致该项目会根据设置的比率占用伸缩容器的额外空间。如果一个伸缩容器里的所有项目都使用此模式，则它们的尺寸会正比于指定的伸缩比率。

默认状态下, 伸缩项目不会收缩至比其最小内容尺寸更小, 可以设置 `min-width` 或 `min-height` 属性来改变这个默认状态。

Flexbox 最强大的特性是能够通过 `flex-flow` 属性设置伸缩项目的流动方向, 或者通过 `flex` 属性设置一个可用的空间。它的取值有三个部分 (`flex-grow`、`flex-shrink` 和 `flex-basis`)。首先添加的是 `flex-grow`。

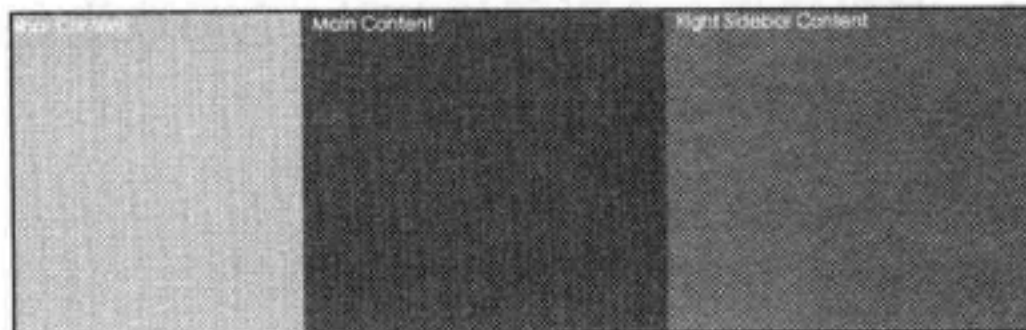
```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>flex 的基本使用 </title>
  <style type="text/css" media="screen">
    *{
      margin: 0;
      padding: 0;
    }
    .flex-container {
      width: 960px;
      margin: 20px auto;
      min-height: 300px;
      color: #fff;
      font-weight: bold;
      display: -webkit-flex;
      display: flex;
      -webkit-flex-flow: row nowrap;
      flex-flow: row nowrap;
      -webkit-justify-content: space-around;
      justify-content: space-around;
      -webkit-align-items: stretch;
      align-items: stretch;
    }
    .flex-container > div {
      padding: 10px;
    }
    .flex-container > div:nth-child(1) {
      background-color: hsla(230,30%,80%,.9);
      -webkit-flex: 1;
      flex: 1;
    }
    .flex-container > div:nth-child(2){
      background-color: hsla(30,80%,10%,.9);
      -webkit-flex: 1;
      flex:1;
    }
    .flex-container > div:nth-child(3){
      background-color: hsla(100,30%,30%,.9);
      -webkit-flex: 1;
      flex:1;
    }
  </style>
</head>
<body>
  <div class="flex-container">
    <div></div>
    <div></div>
    <div></div>
  </div>
</body>
</html>
```



```

</style>
</head>
<body>
  <div class="flex-container">
    <div>Left Sidebar Content</div>
    <div>Main Content</div>
    <div>Right Sidebar Content</div>
  </div>
</body>
</html>

```



其效果如图 8-57 所示。

图 8-57 伸缩项目 flex 取值为 1 的效果

这些没单位的值作为一个比例，它们决定于伸缩容器中有多少个伸缩项目，可以决定伸缩项目在伸缩容器中的空间大小。如果每个都设置为 1，每个伸缩项目在伸缩容器内相等。如果给其中一个伸缩项目设置为 2，这个伸缩项目会占用空间是其他伸缩的两倍。

```

.flex-container > div:nth-child(1) {
  background-color: hsla(230,30%,80%,.9);
  -webkit-flex: 1;
  flex: 1;
}
.flex-container > div:nth-child(2){
  background-color: hsla(30,80%,10%,.9);
  -webkit-flex: 2;
  flex:2;
}
.flex-container > div:nth-child(3){
  background-color: hsla(100,30%,30%,.9);
  -webkit-flex: 1;
  flex:1;
}

```



其效果如图 8-58 所示。

也可以像下面一样设置 flex-basis 的值。

图 8-58 三个伸缩项目的 flex 取值为 1/2/1 的效果

```

.flex-container > div:nth-child(1) {
  background-color: hsla(230,30%,80%,.9);
  -webkit-flex: 1 200px;
  flex: 1 200px;
}
.flex-container > div:nth-child(2){
  background-color: hsla(30,80%,10%,.9);
  -webkit-flex: 2 400px;
  flex:2 400px;
}
.flex-container > div:nth-child(3){
  background-color: hsla(100,30%,30%,.9);
  -webkit-flex: 1 200px;
  flex:1 200px;
}

```

先来看看其效果，如图 8-59 所示。

首先 flex-basis 的值主要取决于伸缩项目的 width 或者 height，同时取决于流动方向；然后，剩下的空间根据 flex-grow 给伸缩项目最后宽度来划分，所以伸缩项目会沿着主轴线大小为 200px、400px、200px。在此例中，每个伸缩项目



图 8-59 伸缩项目设置了 flex-grow 和 flex-basis 的效果

有一个内距 padding 为 10px，这样一来伸缩项目会沿着主轴线大小为 220px、420px 和 220px，总共 860px。如果伸缩容器沿着主轴方向是 960px，这样就多出一个 100px 空间，这多出的 100px 空间将分配给伸缩项目。第一个和第三个伸缩项目将得到 25px 的空间，因为其 flex-grow 值是 1，它们最终的空间是 225px。第二个伸缩项目将获得 50px 的空间，因为其 flex-grow 值为 2，其最后空间大小为 450px。

flex 的第三部分 flex-shrink 很少使用，它称为收缩比率。这个值只有伸缩项目在没主轴方向溢出伸缩容器才会发挥作。它们充当比例值，但这里指的是溢出量，将这个溢出量比例分配给每个伸缩项目，用于防止伸缩容器溢出。

flex 属性在微软的草案与新标准或多或少不一样。主要的区别在于，它们都转换成标准缩写版本，它们的属性值为 flex-grow、flex-shrink 和 flex-basis。然而，flex-shrink（以前称为负 flex）的默认值为 1。这意味着伸缩项目默认不能收缩。以前，空间不足使用 flex-shrink 比例来收缩伸缩项目，但现在可以在 flex-basis 的基础上配合 flex-shrink 来收缩伸缩项目。不过在最旧规范版本中使用的是 box-flex，相对于后面两个版本而言，box-flex 要相对简单，它仅只有一个数值，详细的使用方法可以查阅前面 box-flex 属性的介绍。

8.4.9 显示顺序 order

默认状态下，元素是按照文档流的结构顺序排列。在 Flexbox 模型中，可以通过 order 属性来改变伸缩项目出现在源文档的次序。

```
order: <number>
```

order 属性透过将伸缩项目分到有序号的组以控制伸缩项目的顺序。在伸缩布局中，order 属性控制伸缩项目在伸缩容器里的顺序。伸缩容器会从序号最小的组开始布局，在同一个组里的伸缩项目依据源文档里的次序布局。

在 IE 10 中，这个属性被称为 flex-order，而在更旧的规范版本中是 box-ordinal-group。为了能更好地理解 order 属性的实际使用，一起来看来自 W3C 官网的一个案例。

很多 Web 页面在 HTML 里有很相似的构造，上面有一个页眉，下面有一个页脚，一个内容区块跟一个或两个在中间的额外字段。一般来说，内容出现在页面原始码的前面（在额外的字段之前）比较好。然而，很多一般的设计很难达成，像是把字段摆在内容区块

的两边。多年以来这种俗称“圣杯布局”的两个额外字段的布局已经有很多种方法完成了，然而 `order` 让这种布局轻而易举实现。以下面的页面程序代码草图与预期布局为例，如图 8-60 所示。

结构如下。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>order 属性修改布局顺序 </title>
</head>
<body>
  <header>header</header>
  <section>
    <article>article</article>
    <nav>nav</nav>
    <aside>aside</aside>
  </section>
  <footer>footer</footer>
</body>
</html>
```

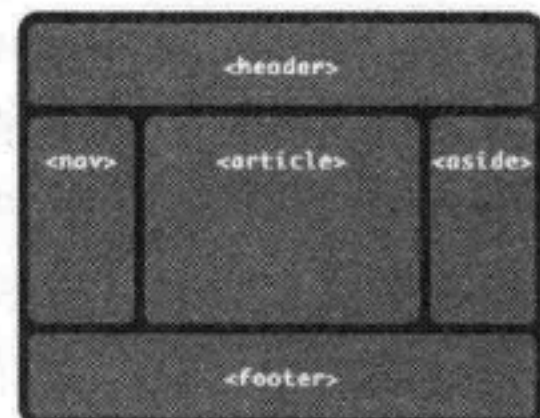


图 8-60 `order` 修改布局顺序

实现图 8-60 布局效果，可以很简单地通过伸缩布局达成。

```
*{
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
html,
body {
  height: 100%;
  color: #fff;
}
body {
  min-width: 100%;
  display: -webkit-flex;
  display: flex;
  -webkit-flex-flow: column wrap;
  flex-flow: column wrap;
  -webkit-justify-content: start;
  justify-content: start;
}
header,
section,
nav,
aside,
footer {
  display: block;
```



```

}
header {
  background-color: hsla(200,10%,70%,.5);
  min-height: 100px;
  padding: 10px 20px;
  width: 100%;
  -webkit-order:1;
  order: 1;
}
section{
  min-width: 100%;
  margin: 20px 0;
  display: -webkit-flex;
  display: flex;
  -webkit-order: 2;
  order: 2;
  -webkit-flex: 1;
  flex:1;
}
nav {
  background-color: hsla(300,60%,20%,.9);
  padding: 1%;
  width: 220px;
  -webkit-order: 1;
  order: 1;
}
article {
  background-color: hsla(120,50%,50%,.9);
  padding: 1%;
  margin-left: 2%;
  margin-right: 2%;
  -webkit-flex: 1;
  flex: 1;
  -webkit-order: 2;
  order: 2;
}
aside {
  background-color: hsla(20,80%,80%,.9);
  padding: 1%;
  width: 220px;
  -webkit-order: 3;
  order: 3;
}
footer {
  background-color: hsla(250,50%,80%,.9);
  min-height: 60px;
  padding: 1%;
  min-width: 100%;
  -webkit-order: 3;
  order: 3;
}

```

在上面的示例中，除了使用 `order` 属性实现了伸缩项目的顺序重组。更棒的是，还使用了 `flex` 属性实现了中间列的等高效果，以及页脚黏附效果，如图 8-61 所示。

8.5 综合案例：跨浏览器的三列布局

Flexbox 模型模块用于布局上绝对是一个强大的工具，也将是未来前端布局的一种主流方案。从浏览器兼容性一节，可以得知，由于 Flexbox 语法规则版本有多个版本，从而也导致浏览器对 Flexbox 模型模块的兼容性也不同。换句话说，到目前为止，各浏览器使用的 Flexbox 语法规则版本不一致。

2009 年开始，一些浏览器（旧的 Webkit 和 Firefox）支持旧版本的 Flexbox 语法规则。2011 年后 IE 10 既支持旧的语法规则版本，又支持新的语法规则版本。实际上它支持介于这两者之间的一种语法规则版本，称为 Flexbox 混合语法规则版本。而在 2011 年 Opera 和 Chrome 首先开始提出支持 Flexbox 标准规范，这也意味着，主流浏览器支持 Flexbox 还需要一段时间。

由于 Flexbox 语法规则版本多，浏览器采用的语法版本不一，以至于使用 Flexbox 就得准备一个方案，让主流浏览器得到完美展示。换一个角度思考，Flexbox 语法规则版本多，主流浏览器采用规范不一，在实际使用中如果把 Flexbox 新语法、旧语法和中间的混合规范版本结合在一起使用，能否让主流浏览器得到一个完美的兼容呢？为了证实这个提意，通过实例来验证。

基于前面的实例，制作一个兼容各主流浏览器的典型的三列布局，其效果如图 8-62 所示。

body 中包括三部分：页眉（header）、主体（section）和页脚（footer）。其中 section 包括左导航栏（nav）、主要内容（article）和右边侧栏（aside）。

```
<header> 我是页眉 </header>
<section>
  <article> 我是主内容 </article>
  <nav> 我是侧边栏导航 </nav>
  <aside> 我是右边栏 </aside>
</section>
<footer> 我是页脚 </footer>
```

实现图 8-62 的布局，需要两步。第一步把 body 和 section 元素变成伸缩容器，此时各

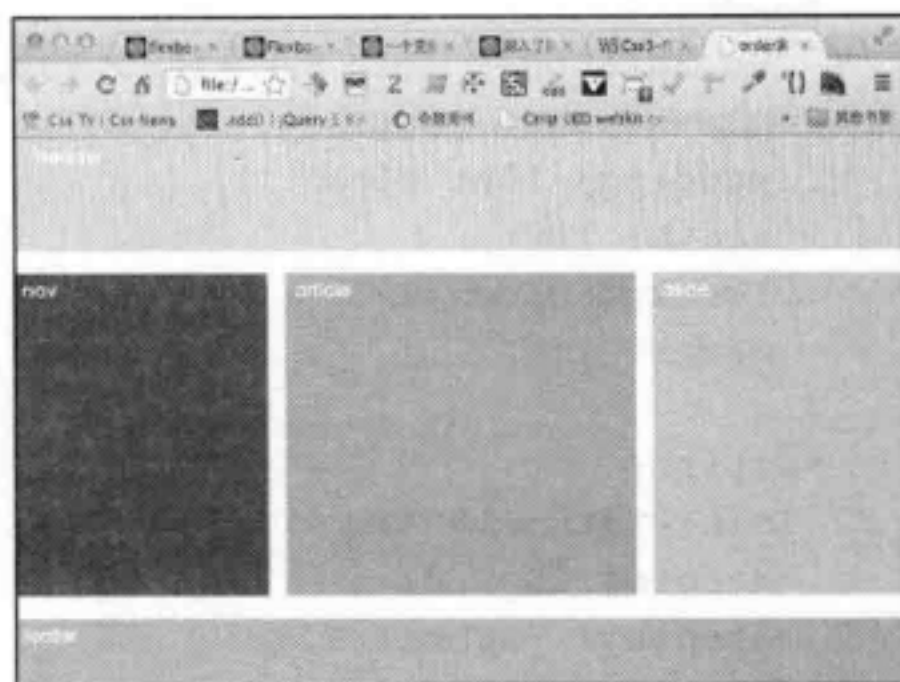


图 8-61 order 重组布局顺序

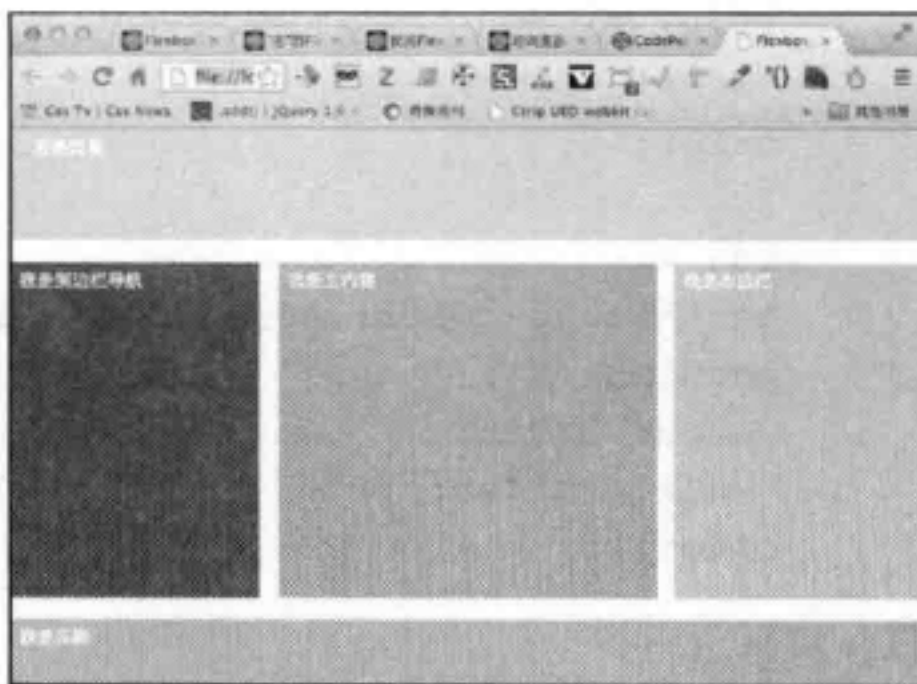


图 8-62 Flexbox 实现三列等高布局

自容器里的每个区域（子元素）自动变成了伸缩项目。需要把 Flexbox 旧的版本、混合版本和最新版本语法混合在一起使用，在这里它们的顺序显得非常重要。

`display` 属性本身并不添加任何浏览器前缀，需要确保旧语法不要覆盖新语法让浏览器（可能总是会）同时支持。

```
/* 设置 body 为伸缩容器 */
body {
  display: -webkit-box; /* 旧版本: iOS 6-, Safari 3.1-6 */
  display: -moz-box; /* 旧版本: Firefox 19- */
  display: -ms-flexbox; /* 混合版本: IE10 */
  display: -webkit-flex; /* 新版本: Chrome */
  display: flex; /* 标准规范: Opera 12.1, Firefox 20+ */
}
/* 设置 section 为伸缩容器 */
section {
  display: -webkit-box; /* 旧版本: iOS 6-, Safari 3.1-6 */
  display: -moz-box; /* 旧版本: Firefox 19- */
  display: -ms-flexbox; /* 混合版本: IE10 */
  display: -webkit-flex; /* 新版本: Chrome */
  display: flex; /* 标准规范: Opera 12.1, Firefox 20+ */
}
```

这时，`body` 和 `section` 设置为伸缩容器，其各自的子元素也自动变成伸缩项目。为了实现图 8-62 的布局，需要把页眉、主体、页脚按文档流顺序垂直排列，并占据整个容器的宽度。实现这个效果，只要在 `body` 容器中设置如下。

```
body {
  /* 设置 body 为伸缩容器 */
  display: -webkit-box; /* 旧版本: iOS 6-, Safari 3.1-6 */
  display: -moz-box; /* 旧版本: Firefox 19- */
  display: -ms-flexbox; /* 混合版本: IE10 */
  display: -webkit-flex; /* 新版本: Chrome */
  display: flex; /* 标准规范: Opera 12.1, Firefox 20+ */

  /* 伸缩项目换行 */
  /* 旧版本: 设置伸缩流 */
  -moz-box-orient: vertical;
  -webkit-box-orient: vertical;
  -moz-box-direction: normal;
  -ms-flex-direction: column;
  /* 旧版本: 伸缩项目换行 */
  -moz-box-lines: multiple;
  -webkit-box-lines: multiple;
  -webkit-flex-flow: column wrap;
  /* 混合版本: 伸缩流方向与伸缩项目换行 */
  -ms-flex-flow: column wrap;
  /* 最新版本: 伸缩流方向与伸缩项目换行 */
  flex-flow: column wrap;
}
```


完成了垂直方向的伸缩项目的布局，接下来需要实现另一个目标，实现三列水平布局。需要在 `section` 伸缩容器中设置伸缩流的方向。

```
section {
  display: -moz-box;
  display: -webkit-box;
  display: -ms-flexbox;
  display: -webkit-flex;
  display: flex;
  /* 设置伸缩流方向 */
  /* 旧版本：设置伸缩流方向 */
  -moz-box-orient: horizontal;
  -webkit-box-orient: horizontal;
  -moz-box-direction: normal;
  -webkit-box-direction: normal;
  /* 旧版本：设置伸缩项目换行 */
  -moz-box-lines: multiple;
  -webkit-box-lines: multiple;
  /* 混合版本：设置伸缩流方向和伸缩项目换行 */
  -ms-flex-flow: row wrap;
  /* 最新版本：设置伸缩流方向和伸缩项目换行 */
  -webkit-flex-flow: row wrap;
  flex-flow: row wrap;
}
```

接下来，需要实现的是 `stick footer` 效果以及让主内容 `article` 自适应伸缩容器。根据前面的实例，要实现 `stick footer` 效果，需要让页面的主体区域能根据视窗高度自适应，也就是要让 `section` 具有伸缩性。这个时候可能给其设置一个伸缩性属性。

```
section {
  display: -moz-box;
  display: -webkit-box;
  display: -ms-flexbox;
  display: -webkit-flex;
  display: flex;
  -moz-box-orient: horizontal;
  -webkit-box-orient: horizontal;
  -moz-box-direction: normal;
  -webkit-box-direction: normal;
  -moz-box-lines: multiple;
  -webkit-box-lines: multiple;
  -ms-flex-flow: row wrap;
  -webkit-flex-flow: row wrap;
  flex-flow: row wrap;
  /* 设置伸缩性 */
  -webkit-box-flex: 1;
  -moz-box-flex: 1;
  -ms-flex: 1;
  -webkit-flex: 1;
}
```

```

    flex: 1;
}

```

按同样的方式给主内容设置一个伸缩性，可以让 article 自适应宽度。

```

article {
    -moz-box-flex: 1;
    -webkit-box-flex: 1;
    -ms-flex: 1;
    -webkit-flex: 1;
    flex: 1;
}

```

到这个时候，其实效果基本上都实现了。希望主内容排列在中间，但在源码之中它是排列在第一的位置。使用 Flexbox 可以非常容易实现，但是需要把 Flexbox 几个不同的语法混在一起使用。

```

article {
    -moz-box-flex: 1;
    -webkit-box-flex: 1;
    -ms-flex: 1;
    -webkit-flex: 1;
    flex: 1;
    /* 旧版本：重排列的顺序 */
    -moz-box-ordinal-group: 2;
    -webkit-box-ordinal-group: 2;
    /* 混合版本：重排列的顺序 */
    -ms-flex-order: 2;
    /* 最新版本：重排列的顺序 */
    -webkit-order: 2;
    order: 2;
}
aside {
    -moz-box-ordinal-group: 3;
    -webkit-box-ordinal-group: 3;
    -ms-flex-order: 3;
    -webkit-order: 3;
    order: 3;
}

```

为了实现中间主内容区域三列等高效果，需要在 section 区域设置侧轴的排列方式。

```

section {
    display: -moz-box;
    display: -webkit-box;
    display: -ms-flexbox;
    display: -webkit-flex;
    display: flex;

    -webkit-box-flex: 1;
}

```

```
-moz-box-flex: 1;
-ms-flex: 1;
-webkit-flex: 1;
flex: 1;

-moz-box-orient: horizontal;
-webkit-box-orient: horizontal;
-moz-box-direction: normal;
-webkit-box-direction: normal;
-moz-box-lines: multiple;
-webkit-box-lines: multiple;
-ms-flex-flow: row wrap;
-webkit-flex-flow: row wrap;
flex-flow: row wrap;
/* 侧轴排列方式 */
-moz-box-align: stretch;
-webkit-box-align: stretch;
-ms-flex-align: stretch;
-webkit-align-items: stretch;
align-items: stretch;
}
```

这样就完成了一个常见的三列等高布局效果，是使用伸缩布局实现，并且兼容所有现代浏览器。正如图 8-62 所示，示例中所有的代码见随书文件。

8.6 本章小结

本章节主要介绍 CSS3 的伸缩布局模块，比起 CSS2.1 所提供的布局技术，伸缩布局模块所创建的布局在处理对齐和间距等问题上具有更强大的能力。

如果使用这个布局模型，只需要将一个容器的 `display` 属性设置成伸缩性容器。接下来就可以使用一系列的新属性来控制伸缩容器内的子元素的排列布局方式。

本章详细介绍了 Flexbox 模型布局功能、属性术语、语法规则版本、浏览器兼容性以及各种不同语法规则版本属性的使用等，并通过实际详细介绍了伸缩属性各种语法规则下的使用方法以及需要注意的细节。通过本章节的学习，大家可以使用伸缩布局模型解决布局问题。

CSS3 多列布局

布局对于每一位 Web 设计师来说都并不陌生。对于布局来说，无外乎是使用浮动 float、行内块 inline-block、定位 position 几种方式，但这几种布局方式都有其不足之处。比如，实现报纸或杂志这样多列布局，几乎不可能使用 CSS 和 HTML 实现。或许会将一篇文章分解为多个 div，让每个 div 浮动，使其看起来像杂志一样的多列布局。但如果内容是动态的，实现起来就很困难，程序员需要判断每列应该在何处开始和结束，以便插入需要的 div 标签。

9.1 CSS3 多列布局简介

CSS3 新增了多列布局特性，可以让浏览器确定何时结束一列和开始下一列，无需任何额外的标记。简单来说，就是 CSS3 多列布局可以自动将内容按指定的列数排列，这种特性实现的布局效果和报纸、杂志类排版非常相似。本章将详细介绍 CSS3 的多列布局的基本属性、用法和实战技巧。

9.1.1 浏览器兼容性

CSS3 中的多列布局包含 columns（column-width 和 column-count 的缩写）、column-gap、column-span 和 column-fill 等属性。目前为止，每个属性在浏览器的兼容性都不太一致，例如其中的 column-span 在 Firefox 中就没有得到友好的支持，而 column-fill 仅在 Firefox 浏览器中得到支持。在实际中使用之时，对于 Webkit 内核浏览和 Firefox 浏览器需要增添它们自己的前缀“-webkit-”和“-moz-”。CSS3 多列布局在浏览器中的兼容性，如图 9-1 所示。

Show all versions	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	BlackBerry Browser
								2.1	
								webkit-2.2	
								webkit-2.3	
						3.2		webkit-3.0	
						4.0-4.1		webkit-4.0	
						4.2-4.3		webkit-4.1	
	8.0					5.0-5.1		webkit-4.2	7.0
	9.0	19.0	-moz- 25.0	-webkit- 5.1	-webkit-	6.0	5.0-7.0	webkit-10.0	-webkit-
Current	10.0	20.0	-moz- 26.0	-webkit- 6.0	-webkit- 12.1				
Near future		21.0	-moz- 27.0	-webkit-					
Farther future		22.0	-moz- 28.0	-webkit-					

☆图 9-1 CSS3 多列布局在浏览器中的兼容性

9.1.2 CSS3 多列布局的属性

多列布局核心的属性包括以下。

- ❑ columns: 集成 column-width 和 column-count 两个属性, 用于实现元素多列布局效果。
- ❑ column-width: 定义每列列宽度。
- ❑ column-count: 定义分列列数。
- ❑ column-gap: 定义列间距。
- ❑ column-rule: 定义列边框。
- ❑ column-span: 定义多布布局中子元素跨列效果。
- ❑ column-fill: 控制每列的列高效果。

CSS3 中多列属性中还有三种属性可供开发人员用于定义分列符应该出现在何处。break-before、break-after 和 break-inside 属性接受有限数量的关键字作为值, 定义分列符应该在元素之前、之后还是内部存在。它们的功能和用法与 CSS2.1 规范中的 page-break-before、page-break-after 和 page-break-inside 三个属性相同。

在 CSS2.1 中 page-break-before、page-break-after 和 page-break-inside 属性具有的 auto、always、avoid、left 和 right 等属性值同样可以用于 break-before、break-after 和 break-inside 属性中, 而且在 CSS3 中还这三个属性增加了 page、column、avoid-page 和 avoid-column 四个属性值。

CSS3 中分列符具有相同的属性值, 其取值说明如下。

- ❑ auto: 不强迫也不禁止在生成框之前(之后、之间)分页。
- ❑ always: 总是强迫在生成框之前(之后)分页。
- ❑ avoid: 避免在生成框之前(之后、之间)分页。
- ❑ left: 强迫在生成框之前(之后)分一个或两个页, 使下一页成为一个左页。
- ❑ right: 强迫在生成框之前(之后)分一个或两个页, 使下一页成为一个右页。
- ❑ page: 总是强迫在生成框之前(之后)分页。

- ❑ column: 总是强迫在生成框之前（之后）分列。
- ❑ avoid-page: 总是避免在生成框之前（之后）分页。
- ❑ avoid-column: 总是避免在生成框之前（之后）分列。

其中 page 和 column 值的作用类似于 always，将强制分列，区别在于 page 仅强制分页，column 仅应用于列，这在换行上提供了更高的灵活性。avoid-page 和 avoid-column 类似，它们的作用类似于 avoid。

9.2 CSS3 多列布局基本属性

为了能在 Web 页面中方便实现类似报纸、杂志多列排版的布局，W3C 特意给 CSS3 增加了一个多列布局模块 (Multiple Column Layout Module)，主要应用在文本的多列布局方面。对于文本的多列布局，大家并不陌生，这种布局在报纸和杂志上都使用了几十年了，如图 9-2 所示。

但在 Web 页面上实现这样的效果还是有相当大的难度。庆幸的是，CSS3 的多列布局可以轻松帮助大家，接下来和大家一起探讨 CSS3 的多列布局的使用方法和注意事项。



图 9-2 报纸和杂志上的多列布局

9.2.1 columns 属性的语法及参数

columns 是 CSS3 中多列布局模块中的一个基本属性，它是一个复合属性——列宽 (column-width) 和列数 (column-count)。此属性类似于前面介绍的 Outline 属性，可以同时定义多列的列数和每列的列宽。columns 属性的基本语法如下所示。

```
columns: <column-width> || <column-count>
```






多列布局 columns 属性参数很简单，主要就两个属性参数，一个是列宽、一个是列数。

- ❑ <column-width>: 定义多列中每列的宽度，详细使用方法请参阅 9.3 节。
- ❑ <column-count>: 定义多列中的列数，详细使用方法请参阅 9.4 节。

9.2.2 浏览器兼容性

在编写本书之时，CSS3 多列布局 columns 属性已得到众多现代浏览器的支持，如表 9-1 所示。

表 9-1 columns 浏览器兼容性

属性名					
columns	10+ ✓	19.0+ ✓	25.0+ ✓	12.1+ ✓	5.1+ ✓

目前为止, 仅有 IE 10+ 和 Opera 12.1+ 浏览器支持标准语法, 而 Firefox 19.0+、Chrome 25.0 + 和 Safari 5.1+ 浏览器还是需要添加浏览器各自的前缀。

9.2.3 实战体验: Web 页面的多列布局

早期在 Web 页面中实现类似的排版多数依赖于浮动, 事先需要计算好使用几个标签, 每个标签中放置多少内容, 本例需要三个 div, 在每个 div 中放置一定数量的内容。效果虽然能实现, 可是各列的内容无法互通, 也难以维护。如图 9-3 所示。



图 9-3 Web 页面的多列布局

使用 CSS3 的多列 columns 属性, 实现这样的布局就不再是问题, 可以轻松地实现。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title> 多列布局 </title>
  <style type="text/css">
    *{
      margin: 0;
      padding: 0;
    }
    .wrapper {
      width: 40em;
      margin-left: auto;
      margin-right: auto;
      border: 1px solid #ccc;
      padding: 5px;
      /* 多列布局 */
    }
  </style>
</head>
```

```

    -moz-columns: 9em 4;
    -webkit-columns: 9em 4;
    columns: 9em 4;
}
h1 {
    font-size: 1.5em;
    margin-bottom: 1em;
}
p {
    margin-bottom: 1em;
    text-indent: 2em;
    line-height: 1.625;
    font-size: .7em;
}

</style>
</head>
<body>
    <div class="wrapper">
        <h1> 金融时报：微软能否拯救 Windows 8？ </h1>
        <p> 网易科技讯。。。： </p>
        <h3> 微软能否拯救 Windows 8？ </h3>
        <p> 这个问。。。 </p>
        <p> 不过，看起。。。 </p>
        <p> 设计融合 PC 和平板。。。 </p>
        <p> 不过微软现在。。。 </p>
        <p> 这些反馈表明，很多用。。。 </p>
        <p> 不过值得微。。。 </p>
    </div>
</body>
</html>

```

其中每列的列宽为 9em，并且分成 4 列。

9.3 CSS3 多列布局列宽属性

多列布局列宽 column-width 属性类似于给列定义一个最小宽度 (min-width)。

9.3.1 column-width 属性的语法及参数

column-width 的使用和 CSS 中的 width 属性一样，不同的是，column-width 属性在定义元素列宽的时候，既可以单独使用，又可以和多列属性中其他属性配合使用。其基本语法如下所示。

```
column-width: auto | <length>
```

column-width 属性取值非常简单。

- **auto**：默认值。如果值为 **auto** 或者没有显式设置值时，元素多列的列宽将由其他属性来决定，比如前面示例就是由列数 **column-count** 来决定。
- **<length>**：使用固定值来设置元素列的宽度，其主要是由数值和长度单位组成，不过其值只能是正值，不能为负值。

9.3.2 实战体验：浏览器根据窗口宽度变化调整列数

首先看一个普通的例子，元素分列的列宽显式设置值，并且其他参数都为默认值。

```
.wrapper {
  /* 多列布局 */
  -moz-column-width: auto;
  -webkit-column-width: auto;
  column-width: auto;
}
```

这个时候元素列的宽度等于自身的宽度，也就是只显示为一列如图 9-4 所示。

金融时报：微软能否拯救Windows 8?

网易科技讯 5月8日消息，据国外媒体报道，Windows 8上市以来遭到业界与用户的各种诟病，销售量也不乐观，虽然微软已经意识到问题的严重性，但是鲍尔默能否力挽狂澜，还不得而知。今天金融时报撰文分析Windows 8与微软所遇到的困境，并展望了他们的未来。以下是文章主要内容：

微软能否拯救Windows 8?

这个问题关乎世界上最大的软件公司的未来以及其首席执行官史蒂夫·鲍尔默 (Steve Ballmer) 的职业生涯。微软为了保持其在PC领域的领先地位而推出的操作系统并没有得到用户的认可。

不过，看起来微软已经意识到问题的严重性。一个标志性动作就是，本周早些时候，微软宣布在今年年底之前发布Windows 8的更新版本，代号为“Blue”。这是微软第一次打破了按年发布的传统，增加软件版本发布的频率，但跟竞争对手苹果和谷歌的软件更新发布频率还有不小差距。不过不管怎样这都说明微软已经开始采取行动，应对更激烈的竞争。

设计融合PC和平板电脑的Win8已经遭到了业界各种严厉的批评。纽约高科技研究公司Envisioneering的分析师Richard Doherty表示，“坐在Win8前面会让人感觉这操作系统蠢毙了，早期使用反馈研究表明人们广泛表示对win8不满。”Win8结合了传统PC界面和平板触屏界面的设计理念也遭人诟病，一些评论认为微软过于雄心勃勃，以至于在两款平台上都没有做好。

不过微软现在已经准备承认该软件确实给PC用户带来了不便：PC用户面对Win8时，需要经历一个十分陡峭的“学习曲线”。同时，微软也在做出改变，旨在提高Win8的可用性，虽然没有确凿证据，但微软受到了来自PC用户不小的压力。PC用户的希望是，当他们打开自己的PC时，是熟悉的Windows桌面，而不是被强制使用“丰富多彩”的微软瓷砖，同时，PC用户也一直希望微软把“开始”按钮弄回来。长期以来这一直是PC用户最熟悉的导航工具。

这些反馈表明，很多用户还没有准备好使用Windows的触屏屏操作系统，而是继续把Windows当作PC时代的产品。微软本周也承认Win8的成功将不仅仅需要用户界面的变化。Win8的另一个问题是价格。不过微软目前正在推动硬件厂商生产尺寸更小，成本更低的平板。这样的话，Win8平板的价格可能迅速下降到400美元左右，并向300美元看齐。

不过值得微软庆幸的是至少目前Win8没有遇到Vista的软件质量问题。许多企业用户因为Vista的各种弊病，直接跳过该版本，直接从早期的XP升级到Windows 7。约三分之二的企业用户目前已经升级到Windows 7。不过他们目前又开始了另一次观望，已决定是否要做新的升级。如果企业用户决定跳过Win8，那么微软将错过追赶苹果和谷歌的重要机会。

图 9-4 column-width 值为 auto 时的效果

给元素设置 **auto** 没有任何效果，因为此时的值需要根据元素多列的其他属性来决定，在没有显式设置的情况下，都将默认显示为一列。

在上面实例的基础上，给元素加上一个 **column-count** 的列数。

```
.wrapper {
  /* 多列布局 */
  -moz-column-width: auto;
```



```

-webkit-column-width: auto;
column-width: auto;
-moz-column-count: 3;
-webkit-column-count: 3;
column-count: 3;
}

```

设置 column-count 属值为 3 时, 元素将内容分成三列, 如图 9-5 所示。



图 9-5 column-width 取值为 auto 并设置 column-count 值为 3 的分列效果

这个再次说明, 当 column-width 属性值为 auto 时需要配合多列的其他属性才能有分列效果。

前面实例给大家演示了 column-width 属性配合其他多列布局属性, 比如设置列数、列间距实现多列布局效果; 除此之外, column-width 还可以单独使用——设置列宽实现多列布局效果。只不过这种方式实现的多列布局效果有些特殊: 当容器超出列宽时, 会以多列显示, 反之容器宽度小于设置的列宽时, 容器将只显示为一列。

在前面的实例上稍做修改, 将容器总宽度设置为 body 的 50% 宽, 同时设置 column-width 的值为 9em。当容器宽度大于列宽, 并且有足够多内容时, 元素多列显示, 反之当容器宽度小于列宽, 元素会以一列显示。

```

<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title> 多列布局 | column-count</title>
  <style type="text/css">
    *{

```

```

    margin: 0;
    padding: 0;
}
.wrapper {
    width: 50%;
    margin-left: auto;
    margin-right: auto;
    border: 1px solid #ccc;
    padding: 5px;
    /* 多列布局 */
    -moz-column-width: 9em;
    -webkit-column-width: 9em;
    column-width: 9em;
}
h1 {
    font-size: 1.5em;
    margin-bottom: 1em;
}
p {
    margin-bottom: 1em;
    text-indent: 2em;
    line-height: 1.625;
    font-size: .7em;
}

</style>
</head>
<body>
    <div class="wrapper">
        <h1> 金融时报：微软能否拯救 Windows 8？ </h1>
        <p> 网易科...： </p>
        <h3> 微软能否拯救 Windows 8？ </h3>
        <p> 这个问题... </p>
        <p> 不过，看起来微软已... </p>
        <p> 设计融合... </p>
        <p> 不过微软现在... </p>
        <p> 这些反馈表明... </p>
        <p> 不过值得微... </p>
    </div>
</body>
</html>

```

随着浏览器窗口大小变化，容器的宽度大小也变化，容器的列数也不断的变化，演示效果如图 9-6 所示。

上例说明在容器本身太窄的时候，无法容纳指定宽度的一列，将由一列填满整个容器。

另外一种情形是，当容器没有足够空间来包含具有指定宽度的列数，元素列数会自动往下计算。例如，容器的总宽度为 40em，列间距为 2em，通过 column-width 显式设置每列的列宽为 10em，根据容器宽度、列宽度、间距和列数之间的关系



图 9-6 浏览器根据窗口宽度变化调整列数

$$(\text{容器宽度} - \text{列与列间距}) / \text{列宽} = \text{列数}$$

可以很快算数列数：

$$(40\text{em} - 2\text{em}) / 10\text{em} = 38\text{em} / 10\text{em} = 3.8$$

根据浏览器的四舍五入原则，此时的列数应该是 4，根据所计算的列数运用到实例中。

```
.wrapper {
  /* 多列布局 */
  -moz-column-width: 10em;
  -webkit-column-width: 10em;
  column-width: 10em;
  -moz-column-gap: 2em;
  -webkit-column-gap: 2em;
  column-gap: 2em;
  -moz-column-count: 4;
  -webkit-column-count: 4;
  column-count: 4;
}
```

演示效果，如图 9-7 所示。

虽然按计算，容器应该分为四列，同时也在容器中显式的设置了列数为四列，但实际效果却只显示了三列。造成这种现象的主要原因是容器没有足够宽的空间来包含具有指定宽度的四列。换句话说，可以认为 column-count 属性指定了最大列数。

金融时报：微软能否拯救Windows 8?

网易科技讯 5月8日消息，据国外媒体报道，Windows 8上市以来遭到业界与用户的各种诟病，销量也不乐观。虽然微软已经意识到问题的严重性，但是鲍尔默能否力挽狂澜，还不得而知。今天金融时报撰文分析Windows 8与微软所遇到的困境，并展望了他们的未来。以下是文章主要内容：

微软能否拯救Windows 8?

这个问题关乎世界上最大的软件公司的未来以及其首席执行官史蒂夫·鲍尔默 (Steve Ballmer) 的职业生涯。微软为了保持其在PC领域的领先地位而推出的操作系统，没有得到用户的认可。

不过，看起来微软已经意识到了问题的严重性。一个标志性的事件是，微软在本周早些时候，宣布将放弃之前发布Windows 8的更新版本，代

号为“Blue”。这是微软第一次打破了按年发布的传统，增加软件版本发布的频率，但竞争对手苹果和谷歌的软件更新发布频率还有不小差距。不过不管怎样这都说明微软已经开始采取行动，应对更激烈的竞争。

设计融合PC和平板电脑的Win8已经遭到了业界各种严厉批评。纽约高科技研究公司Envisioneering的分析师Richard Doherty表示，“坐在Win8前面会让人感觉这操作系统蠢透了，早期使用反馈研究表明人们广泛表示对win8不满。” Win8结合了传统PC界面和平板触摸屏界面的设计理念也遭到诟病，一些评论认为微软过于雄心勃勃，以至于在两款平台上都没有做好。

不过微软现在已经准备承认该软件确实给PC用户带来了不便：PC用户对Win8时，需要经历一个十分陡峭的学习曲线。同时，微软也意识到，旨在提高Win8的可用性，虽然没有确凿证据，但微软受到了来自用户不小的压力。PC用户时，是熟悉的Windows桌面，而不

是被强制使用“丰富多彩”的微软瓷砖。同时，PC用户也一直希望微软把“开始”按钮弄回来。长期以来这一直是PC用户最熟悉的导航工具。

这些反馈表明，很多用户还没有准备好使用Windows的触摸屏操作系统，而是继续把Windows当作PC时代的产品。微软本周也承认Win8的成功将不仅仅需要用户界面的变化。Win8的另一个问题是价格。不过微软目前正在推动硬件厂商生产尺寸更小，成本更低的平板。这样的话，Win8平板的价格可能迅速下降到400美元左右，并向300美元看齐。

不过值得微软庆幸的是至少目前Win8没有遇到Vista的软件质量问题。许多企业用户因为Vista的各种弊病，直接跳过该版本，直接从早期的XP升级到Windows 7。约三分之二的企业用户目前已经升级到Windows 7。不过他们目前又开始了另一次观望，已决定是否要做新的升级。如果企业用户决定跳过，那么微软将错过追赶苹果和谷歌的重要机会。

容器宽度: 40em
列宽: 10em
列间距: 2em
列数 = (40em - 2em) / 10em = 3.8

图 9-7 最大列数的分列效果

9.4 CSS3 多列布局列数属性

column-count 属性主要用来给元素指定想要的列数和允许的最大列数。

9.4.1 column-count 属性的语法及参数

column-count 属性的语法和 column-width 一样，也非常简单。

```
column-count: auto | <integer>
```

column-count 主要有两种取值方式。

- auto: 默认值，表示元素只有一列。其主要依靠浏览器计算自动设置。
- <integer>: 正整数值，主要用来定义元素的列数，取值为大于 0 的整数，负值无效。

9.4.2 实战体验：显示固定列数

column-count 的列数计算其实在前面的 column-width 的计算中有提到，根据元素的宽度、列宽以及间距，就可以轻松地计算出元素分列的列数。

列数 = (容器宽度 - 间距) / 列间距

不过这里有一个关键的地方，因为计算的值并不一定每次都是正整数，总是会有小数

位出现。浏览器对于小数的取舍正常情况之下都是四舍五入，但在这里并不是这样，对于列数的取值，是去余取整。也就是：

$$\text{列数} = (40\text{em} - 2\text{em}) / 10\text{em} = 38\text{em} / 10\text{em} = 3.8$$

按照浏览器的取值习惯，列数四舍五入后，其列数应该是4列，但在实际中并不是这样，就算是在实例中显式的设置了列数为4列，浏览器最终解析出来也不是4列，而是3列。这样一来，column-count 在实际中取值只是去余取整，在这里也就是3列。

另外就是，如果元素显式设置了列数，不管元素容器的宽度是多少或者间距多少，总是显示固定的列数，如图9-8所示。



图 9-8 固定列数显示效果

代码如下。

```
.wrapper {
  /* 多列布局 */
  -moz-column-count: 4;
  -webkit-column-count: 4;
  column-count: 4;
}
```

9.5 CSS3 多列布局列间距属性

column-width 和 column-count 可以很方便地让一个元素进行多列布局，从前面的实例

中也简单地提到了列与列的间距，其实这个间距就是接下来要介绍的列属性 `column-gap`。用印刷排版的术语来说，`column-gap` 就相当于两列之间的空白此处。换句话说，`column-gap` 就像 Web 页面中的 `margin` 一样，不过只是存在相邻两列之间。

9.5.1 `column-gap` 属性的语法及参数

`column-gap` 类似于盒模型中的 `margin` 一样，主要用来设置元素分列的列间距，其只能设置列与列之间的间距。其使用语法很简单。

```
column-gap: normal | <length>
```

`column-gap` 主要由关键词 `normal` 或者使用长度值来显式的设置列间距。

- `normal`：默认值，主要通过浏览器默认设置时行解析，一般情况下，`normal` 值相当于 `1em`。
- `<length>`：由浮点数字和单位标识符组成的长度值，主要用来设置列与列之间的距离，常用 `px`、`em` 单位的任何整数值，但其不能为负值。

9.5.2 实战体验：设置列间距

`column-gap` 属性和 `margin` 属性非常类似，主要用来设置列之间的间距。不过 `margin` 属性用于元素与元素之间，不能用于列与列之间；`column-gap` 却和 `margin` 相反，其主要用于元素多列之间间距设置，而不能用于元素与元素之间的设置。

现在基于上节的例子，稍加改动，通过 `column-gap` 显式给元素设置列间距。

```
.wrapper {  
  width: 40em;  
  margin-left: auto;  
  margin-right: auto;  
  border: 1px solid #ccc;  
  padding: 5px;  
  /* 多列布局 */  
  -moz-column-count: 4;  
  -webkit-column-count: 4;  
  column-count: 4;  
  -moz-column-gap: 2em;  
  -webkit-column-gap: 2em;  
  column-gap: 2em;  
}
```

其效果如图 9-9 所示。

上面的实例是在 `column-width` 为 `auto` 下把元素分成 4 列，并显式设置了列间距为 `2em`。从图中明显可以看出，`column-gap` 将相邻两列以 `2em` 的宽度分隔开，整个页面的排版清晰。

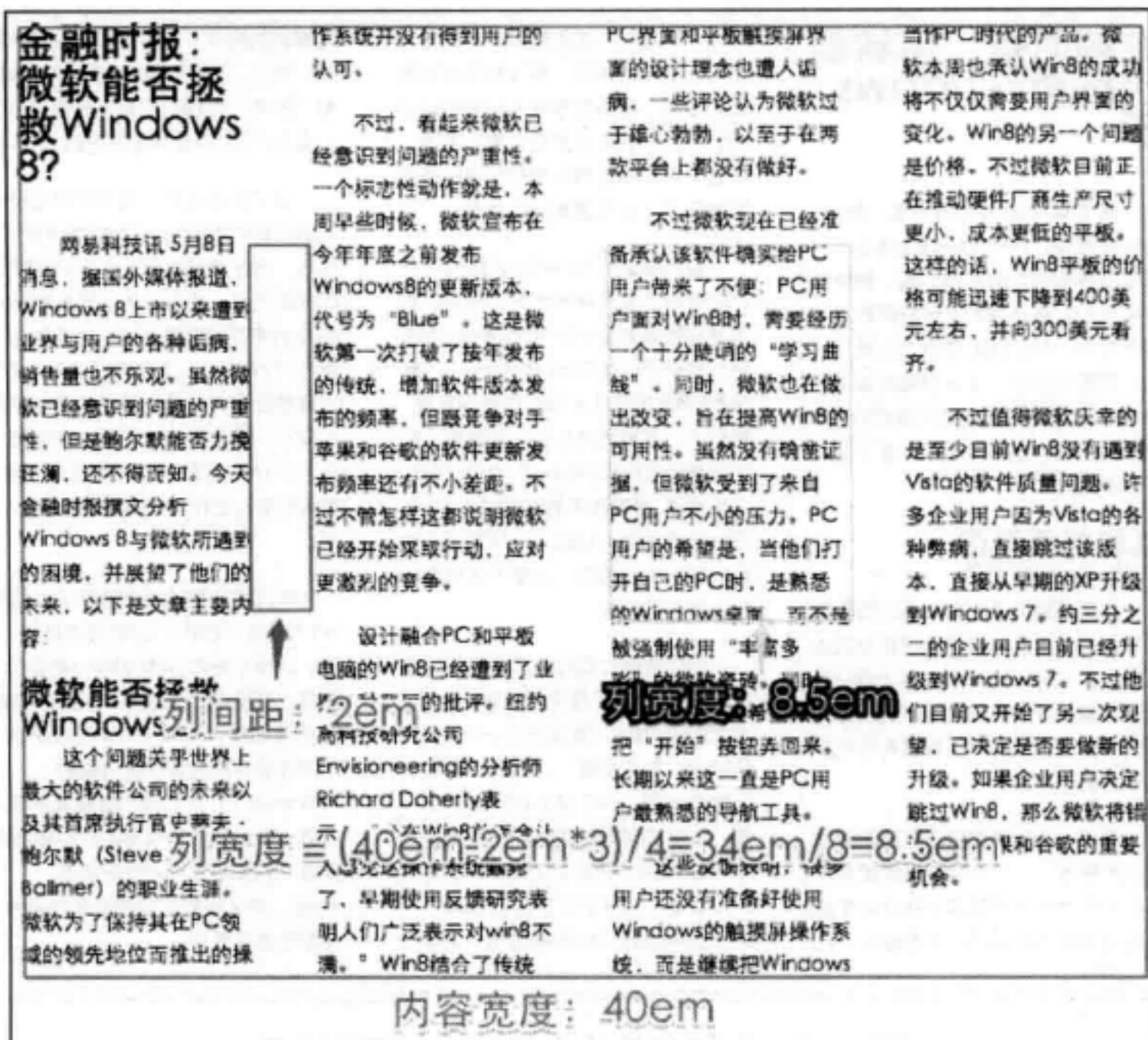


图 9-9 column-gap 设置列间距

根据相关参数，可以轻松算出示例中的每列列宽是 8.5em。如果显式地将列宽度设置为 8.51em，比浏览器自动计算出来的列宽度大 0.01em。

```
.wrapper {
  width: 40em;
  margin-left: auto;
  margin-right: auto;
  border: 1px solid #ccc;
  padding: 5px;
  /* 多列布局 */
  -moz-column-count: 4;
  -webkit-column-count: 4;
  column-count: 4;
  -moz-column-gap: 2em;
  -webkit-column-gap: 2em;
  column-gap: 2em;
  -moz-column-width: 8.51em;
  -webkit-column-width: 8.51em;
  column-width: 8.51em;
}
```

效果如图 9-10 所示。

金融时报：微软能否拯救Windows 8?

网易科技讯 5月8日消息，据国外媒体报道，Windows 8上市以来遭到业界与用户的各种诟病，销量也不乐观。虽然微软已经意识到问题的严重性，但是鲍尔默能否力挽狂澜，还不得而知。今天金融时报撰文分析Windows 8与微软所遇到的困境，并展望了他们的未来，以下是文章主要内容：

微软能否拯救Windows 8?

这个问题关乎世界上最大的软件公司的未来以及其首席执行官史蒂夫·鲍尔默 (Steve Ballmer) 的职业生涯。微软为了保持其在PC领域的领先地位而推出的操作系统并没有得到用户的认可。

不过，看起来微软已经意识到问题的严重性。一个标志性动作就是，本周早些时候，微软宣布在今年年底之前发布Windows 8的更新版本，代

号为“Blue”。这是微软第一次打破了按年发布的传统，增加软件版本发布的频率，但跟竞争对手苹果和谷歌的软件更新发布频率还有不小差距。不过不管怎样这都说明微软已经开始采取行动，应对更激烈的竞争。

设计融合PC和平板电脑的Win8已经遭到了业界各种严厉的批评。纽约高科技研究公司Envisioneering的分析师Richard Doherty表示，“坐在Win8前面会让人感觉这操作系统蠢毙了，早期使用反馈研究表明人们广泛表示对win8不满。”Win8结合了传统PC界面和平板触摸屏界面的设计理念也遭人诟病，一些评论认为微软过于雄心勃勃，以至于在两款平台上都没有做好。

不过微软现在已经准备承认该软件确实给PC用户带来了不便：PC用户对Win8时，需要经历一个十分陡峭的“学习曲线”。同时，微软也在做出改变，旨在提高Win8的可用性。虽然没有确凿证据，但微软受到了来自PC用户不小的压力。PC用户的希望是，当他们打开自己的PC时，是熟悉的Windows桌面，而不

是被强制使用“丰富多彩”的微软瓷砖。同时，PC用户也一直希望微软把“开始”按钮弄回来。长期以来这一直是PC用户最熟悉的导航工具。

这些反馈表明，很多用户还没有准备好使用Windows的触摸屏操作系统，而是继续把Windows当作PC时代的产品。微软本周也承认Win8的成功将不仅仅需要用户界面的变化。Win8的另一个问题是价格。不过微软目前正在推动硬件厂商生产尺寸更小，成本更低的平板。这样的话，Win8平板的价格可能迅速下降到400美元左右，并向300美元看齐。

不过值得微软庆幸的是至少目前Win8没有遇到Vista的软件质量问题。许多企业用户因为Vista的各种弊病，直接跳过该版本，直接从早期的XP升级到Windows 7。约三分之二的企业用户目前已经升级到Windows 7。不过他们目前又开始了另一次观望，已决定是否要做新的升级。如果企业用户决定跳过Win8，那么微软将错过追赶苹果和谷歌的重要机会。

图 9-10 多列参数值之和超过总宽度效果

上图告诉我们，浏览器无法显示 4 列。大家都知道，元素宽度为 40em，现在每列定在 8.51em 列宽，列间距 2em。这是一个很简单的数学题，多列各参数值之和超过多列元素总宽度，以致元素无法按多列参数值进行布局。也就是说，column-gap 可以用来改变相邻列之间距离，但在多列元素同时设置了 column-width 时，column-gap 与 column-width 等参数之和大于多列元素总宽度时，会导致列被撑破，并将当前列数减 1 的列数显示，此时的列宽自动调节到适当的列宽。换句话说，当列宽度足够大时，以至于无法分列显示时，就算设置了列数、列间距，都将以一列显示，并且会自动调整列宽等于元素的宽度。

9.6 CSS3 多列布局列边框样式属性

column-rule 主要是用来定义列与列之间的边框宽度、边框样式和边框颜色。简单点说，就有点类似于常用的 border 属性。但 column-rule 是不占用任何空间位置的，在列与列之间改变其宽度不会改变任何列的位置。

9.6.1 column-rule 属性的语法及参数

你可以把 column-rule 当作元素中的 border 属性理解。因为 column-rule 和 border 具有类似的属性：边框宽度 column-rule-width、边框样式 column-rule-style 和边框颜色 column-

rule-color。

```
column-rule:<column-rule-width>|<column-rule-style>|<column-rule-color>
```

为了方便设计师灵活的设计列边框，column-rule 属性派生出三个列边框属性。

- ❑ column-rule-width：类似于 border-width 属性，主要用来定义列边框的宽度，其默认值为 medium，该属性接受任意浮点数，但不接受负值。像 border-width 属性一样，可以使用关键词 medium、thick 和 thin。
- ❑ column-rule-style：此值类似于 border-style 属性，主要用来定义列边框样式，其默认值为 none。该属性值与 border-style 属性值相同，包括 none、hidden、dotted、dashed、solid、double、groove、ridge、inset、outset。
- ❑ column-rule-color：此值类似于 border-color 属性，主要用来定义列边框颜色，其默认值为前景色 color 的值，使用时相当于 border-color。该属性接受所有的颜色。如果不希望显示颜色，也可以将其设置为 transparent（透明色）。

9.6.2 实战体验：设置列边框

在前面示例的基础上，为每列之间设置一个边框。下面的示例给每列增添一个 2px 的虚线边框。

```
.wrapper {
  width: 40em;
  margin-left: auto;
  margin-right: auto;
  border: 1px solid #ccc;
  padding: 5px;
  /* 多列布局 */
  -moz-column-count: 4;
  -webkit-column-count: 4;
  column-count: 4;
  -moz-column-gap: 2em;
  -webkit-column-gap: 2em;
  column-gap: 2em;
  -moz-column-rule: 2px dashed #ccc;
  -webkit-column-rule: 2px dashed #ccc;
  column-rule: 2px dashed #ccc;
}
```

效果如图 9-11 所示。

这是一个非常简单的实例，在列与列之间设置一个 2px 宽的灰色虚线边框。在这个示例的基础上，把刚才示例中的列边框宽度变大，超过列间距。

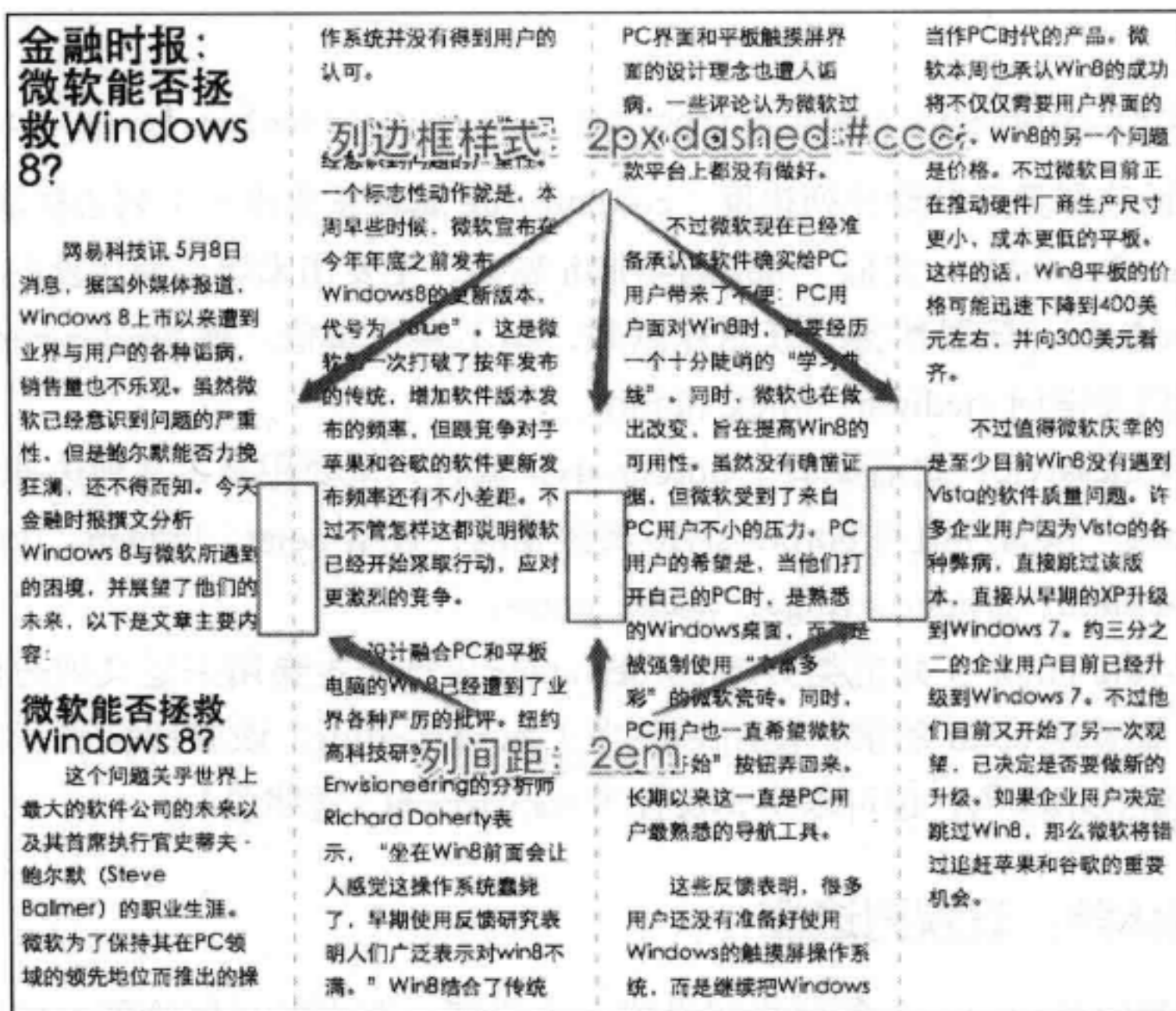


图 9-11 设置列边框效果

```

.wrapper {
  width: 40em;
  margin-left: auto;
  margin-right: auto;
  border: 1px solid #ccc;
  padding: 5px;
  /* 多列布局 */
  -moz-column-count: 4;
  -webkit-column-count: 4;
  column-count: 4;
  -moz-column-gap: 2em;
  -webkit-column-gap: 2em;
  column-gap: 2em;
  -moz-column-rule: 2.1em dashed #ccc;
  -webkit-column-rule: 2.1em dashed #ccc;
  column-rule: 2.1em dashed #ccc;
}

```

此时列边框超过了列间距，列边框自动消失。效果如图 9-12 所示。

图 9-11 效果告诉我们，其一，column-rule-width 增大并不会影响列的布局，也就是说其不占有任何空间位置；其二，column-rule 在 Z 轴上是介于 background 和 content 之间。但 column-rule-width 宽度增加到超过列与列之间距，那么列边框就会自动消失。

金融时报： 微软能否拯救 Windows 8?

网易科技讯 5月8日消息，据国外媒体报道，Windows 8上市以来遭到业界与用户的各种诟病，销售量也不乐观。虽然微软已经意识到问题的严重性，但是鲍尔默能否力挽狂澜，还不得而知。今天金融时报撰文分析 Windows 8与微软所遇到的困境，并展望了他们的未来，以下是文章主要内容：

微软能否拯救 Windows 8?

这个问题关乎世界上最大的软件公司的未来以及其首席执行官史蒂夫·鲍尔默(Steve Ballmer)的职业生涯。微软为了保持其在PC领域的领先地位而推出的操

作系统并没有得到用户的认可。

不过，看起来微软已经意识到问题的严重性。一个标志性动作就是，本周早些时候，微软宣布在今年年底之前发布 Windows 8的更新版本，代号为“Blue”。这是微软第一次打破了按年发布的传统，增加软件版本发布的频率，但跟竞争对手苹果和谷歌的软件更新发布频率还有不小差距。不过不管怎样这都说明微软已经开始采取行动，应对更激烈的竞争。

设计融合PC和平板电脑的Win8已经遭到了业界各种严厉的批评。纽约高科技研究公司 Envisioneering的分析师 Richard Doherty表示，“坐在Win8前面会让人感觉这操作系统蠢毙了，早期使用反馈研究表明人们广泛表示对win8不满。” Win8结合了传统

PC界面和平板触摸屏界面的设计理念也遭人诟病，一些评论认为微软过于雄心勃勃，以至于在两款平台上都没有做好。

不过微软现在已经准备承认该软件确实给PC用户带来了不便：PC用户面对Win8时，需要经历一个十分陡峭的“学习曲线”。同时，微软也在做出改变，旨在提高Win8的可用性。虽然没有确凿证据，但微软受到了来自PC用户不小的压力。PC用户的希望是，当他们打开自己的PC时，是熟悉的Windows桌面，而不是被强制使用“丰富多彩”的微软瓷砖。同时，PC用户也一直希望微软把“开始”按钮弄回来。长期以来这一直是PC用户最熟悉的导航工具。

这些反馈表明，很多用户还没有准备好使用 Windows的触摸屏操作系统，而是继续把Windows

当作PC时代的产品。微软本周也承认Win8的成功将不仅仅需要用户界面的变化。Win8的另一个问题是价格。不过微软目前正在推动硬件厂商生产尺寸更小、成本更低的平板。这样的话，Win8平板的价格可能迅速下降到400美元左右，并向300美元看齐。

不过值得微软庆幸的是至少目前Win8没有遇到 Vista的软件质量问题。许多企业用户因为Vista的各种弊病，直接跳过该版本，直接从早期的XP升级到Windows 7，约三分之二的企业用户目前已经升级到Windows 7。不过他们目前又开始了另一次观望，已决定是否要做新的升级。如果企业用户决定跳过Win8，那么微软将错过追赶苹果和谷歌的重要机会。

图 9-12 列边框宽度超过列间距，列边框消息

扩展阅读 多列列边框与列间距总结

通过前两节对多列列间距和列边框的介绍，最后规纳如下。

- 多列列间距 column-gap 就类似于盒模型中的 margin 和 padding 属性，具有一定的空间位置，当其值过大时也会撑破列布局，浏览器会自动根据相关参数重新计算列数，直到容器无法容纳时，显示为一列为止。但多列列间距 column-width 和 margin、padding 属性不同的是，其只存在列与列之间，并与列高度相等。
- 多列列边框 column-rule 就类似于盒模型中的 border 属性，主要用来设置列边框的宽度、列边框的样式和列边框颜色，并且 column-rule 不具有任何空间位置，同时在 Z 轴介于盒模型 background 和 content 之间，其同样具有与列一样的高度，但列边框 column-rule 和 border 不一样，border 会撑破容器，而 column-rule 不会撑破容器，只不过其列宽宽度大于列间距宽度时，列边框会自动消失。

9.7 CSS3 多列布局跨列属性

报纸或杂志文章的标题经常会跨列显示，但从前面的实例发现，所有示例的分列，标题都在当前列显示。要设置如报纸或杂志上的跨列显示效果，就需要借助于 column-span 属性。

9.7.1 column-span 属性的语法及参数

column-span 主要用来定义一个分列元素中的子元素能跨列多少。column-width、column-count 等属性能让一个元素分成多列，不管里面元素如何排放顺序，它们都是从左向右放置内容，但有时需要其中一段内容或一个标题不进行分列，也就是横跨所有列，此时 column-span 就可以轻松实现，此属性的语法如下。

column-span: none | all

column-span 属性参数非常简单，其取值如下。

- none: 默认值，表示不跨越任何列。
- all: 跟 none 值刚好相反，表示的是元素跨越所有列，并定位在列的 Z 轴之上。

9.7.2 实战体验：文章标题跨列显示

在这个实例中，通过 column-span 属性来实现报纸或杂志中文章标题跨列居中显示效果。只需要在前面的示例基础上，在所有标题中设置一个 column-span 属性。其效果如图 9-13 所示。



图 9-13 column-span 制作标题跨列效果

实现代码如下所示。

```
.wrapper {
  width: 40em;
  margin-left: auto;
```



```

margin-right: auto;
border: 1px solid #ccc;
padding: 5px;
/* 多列布局 */
/* 设置列数 */
-moz-column-count: 4;
-webkit-column-count: 4;
column-count: 4;
/* 设置列间距 */
-moz-column-gap: 2em;
-webkit-column-gap: 2em;
column-gap: 2em;
/* 设置列边框 */
-moz-column-rule: .1em dashed #ccc;
-webkit-column-rule: .1em dashed #ccc;
column-rule: .1em dashed #ccc;
}
h1 {
font-size: 1.5em;
margin-bottom: 1em;
padding-bottom: 5px;
text-align: center;
border-bottom: 2px solid #ccc;
/* 标题跨列 */
-moz-column-span: all;
-webkit-column-span: all;
column-span: all;
}

```

9.8 CSS3 多列布局列高度属性

在列布局当中，有时由于内容不足，在多列之中最后列往往会没有足够内容填充。但实际制作中往往有时候的页面效果就必须让所有列都具有相同的高度效果。这个时候需要这一节将要介绍的属性 `column-fill`。

简而言之，`column-fill` 属性主要是用来定义多列中每一列的高度是否统一。`column-fill` 属性的基本语法。

```
column-fill: auto | balance
```

`column-fill` 只有两个属性值。

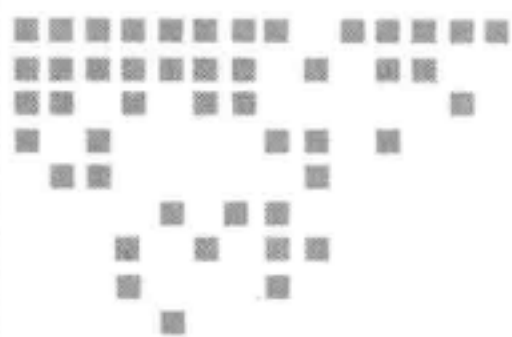
❑ `auto`：默认值，各列的高度随其内容的变化自动变化。

❑ `balance`：各列的高度将会根据内容最多的一列的高度进行统一。

9.9 本章小结

CSS3 的多列布局可以帮我们在 Web 页面中轻松实现类似于报纸或杂志那种排版效果，而无须增添一些无用的标签以及依赖于浮动或者定位来实现那种布局效果。本章把 CSS3 的多列属性拆解分述，详细介绍了每一个属性在实际运用的使用方法以及使用细节。

CSS3 渐变



一直以来，Web 设计师都是通过图形软件设计这些渐变效果，然后以图片形式或者背景图片的形式运用到页面中。Web 页面上实现的效果，仅从页面的视觉效果上来看，与设计并无任何差异。

事实上这种方法比较麻烦，因为首先需要设计师进行设计，然后进行切图，再通过样式应用到页面中。另外，在实际应用中可扩展性差，还直接影响页面性能。

值得庆幸的是，W3C 组织将渐变设计收入到 CSS3 标准中，可以直接通过 CSS3 的渐变属性制作类似渐变图片的效果。

10.1 CSS3 渐变简介

CSS3 渐变是什么？从早前的设计中我们可以知道，渐变是两种或多种颜色之间的平滑过渡。

10.1.1 什么是色标

在创建渐变的过程中，可以指定多个中间颜色值，这个值称为色标。每个色标包含一种颜色和一个位置，浏览器从每个色标的颜色淡出到下一个，以创建平滑的渐变，如图 10-1 所示。

渐变可以应用于任何使用背景图片的地方。在 CSS 样式中，渐变相当于背景图片，在理论上可在任何使用 `url()`



☆图 10-1 渐变与渐变中的色标

值的地方采用, 比如最常见的 background-image、list-style-type 以及前面介绍的 CSS3 图像边框属性 border-image。目前为止, 仅在背景图片中得到完美的支持。

10.1.2 浏览器兼容性

最早支持 CSS3 渐变的是 Webkit 内核的浏览器, 随后在 Firefox 和 Opera 等浏览器得到支持, 但是众浏览器之间没有得到统一的标准, 用法差异很大。不同的渲染引擎实现渐变的语法也不同, 各浏览器下使用都需要带上自己的前缀, 给前端设计师们带来极大的不便。






不过还好, 到写本章内容的时候, CSS3 渐变属性在 IE 10+、Firefox 19.0+、Chrome 26.0+ 和 Opera 12.1+ 等浏览器已完全支持 W3C 的标准语法, 但在 Webkit 内核下的 Safari、iOS Safari、Android 浏览器和 Blackberry 浏览器中还是需要添加浏览器的前缀“-webkit-”。

CSS3 的渐变的语法几经变化, 不过让前端设计师庆幸的是, 直到写本书的时候, CSS3 的渐变语法除了在 Webkit 的 Safari 浏览器和 IE 10 以下的浏览器没有得到支持之外, 其他浏览器对渐变支持都很稳定。



注意 表 10-1 所列的是 W3C 的渐变标准语法在浏览器兼容性。

表 10-1 渐变的浏览器兼容性

属性名					
Gradient	10+ ✓	19.0+ ✓	26.0+ ✓	12.1+ ✓	5.1+ ✓

在介绍 CSS3 渐变属性的时候, 就知道渐变的语法版本种类有多种, 而各语法版本浏览器的兼容性也各有不同, 如表 10-2 所示。

表 10-2 渐变属性的浏览器支持

	线性渐变	径向渐变
超老版本浏览器供应商语法	Chrome 4 ~ 9, Safari 4 ~ 5	Chrome 4 ~ 9, Safari 4 ~ 5
老版本浏览器供应商语法	IE 10+、Firefox 3.6+、Chrome 10+、Safari 5.1+、Opera 11.6+	IE 10+、Firefox 3.6+、Chrome 10+、Safari 5.1+、Opera 11.6
新版本浏览器供应商语法	Firefox 10+、Opera 11.6+	
W3C 标准语法	IE 10+、Firefox 19+、Chrome 26+、Opera 12.1+	IE 10+、Firefox 19+、Chrome 26+、Opera 12.1+

1. IE 低版本浏览器兼容处理

对于 IE 9 以前的版本, 并不支持 CSS3 渐变属性, 但可以使用 IE 的专有滤镜来创建简单的渐变。IE 渐变滤镜不支持色标、渐变角度和径向渐变。我们能做的仅能指定水平和垂直的线性渐变, 以及渐变的开始颜色和渐变的结束颜色。

如果需要为这些比较旧的浏览器提供渐变效果, 可以按照下面的滤镜语法来实现。


```
filter:progid:DXImageTransform.Microsoft.gradient(GradientType=0,
startColorstr='#COLOR',endColorstr='#COLOR');/*IE6 和 IE7*/
-ms-filter:"progid:DXImageTransform.Microsoft.gradient(GradientType=0,
startColorstr='#COLOR',endColorstr='#COLOR')";/*IE8*/
```

语法中的 GradientType 参数主要用来设置渐变的方向，其中值为 1 表示的是水平线性渐变，值为 0 表示的是垂直线性渐变。

startColorstr 参数主要用来设置或检索色彩渐变的开始颜色和透明度。其值是一个可选值。其值“#COLOR”格式为“#AARRGGBB”。AA、RR、GG、BB 为十六进制正整数，取值范围为 00 ~ FF。RR 指定红色值，GG 指定绿色值，BB 指定蓝色值，AA 指定透明度，00 表示完全透明，FF 则表示完全不透明。超出取值范围的值将被恢复为默认值。取值范围为“#FF000000”到“FFFFFFFF”，默认值为“#FF0000FF”。

endColorstr 参数主要用来设置或检索色彩渐变的结束颜色和透明度。其值“#COLOR”的格式和“startColorstr”取值格式一样。默认值为“#FF000000”。

2. 针对其他浏览器的兼容方案

CSS3 渐变在众多现代浏览器中都得到较好的支持，因为它为不兼容的浏览器提供解决方案来。

- ❑ 使用图片。对于不支持 CSS3 渐变特性的浏览器来说，最简单的方法就是按照老方法来实现渐变：创建一个半透明的 PNG 渐变图片。在使用这个解决方案的时候，必须注意一点，运用的图片要写在 CSS 渐变属性之前，只有这样才能让支持渐变的浏览器用创造渐变效果的 background-image 属性覆盖指定真实图片同名规则。
- ❑ 使用脚本。对于 IE 6 ~ 8 兼容 CSS3 的渐变效果，还可以使用 PIE 脚本^①。对于其他浏览器，可以试试 Weston Ruter 的“用 Canvas 模拟 CSS3 渐变”的脚本^②。
- ❑ 采用纯色。对于不兼容的浏览器，设置一个 background-color 色，使其在浏览器中显示为纯色。
- ❑ 使用滤镜。对于 IE 9 以下浏览器兼容 CSS3 的渐变效果，可以采用 IE 专有的滤镜语法。

10.2 CSS3 线性渐变

在线性渐变过程中，颜色沿着一条直线过渡：从左侧到右侧、从右侧到左侧、从顶部到底部、从底部到顶部或沿任意轴。如果使用过制作 Photoshop 等软件，对线性渐变并不会陌生。

CSS3 制作渐变效果，其实和使用制作软件中的渐变工具没有什么差别。首先指定一个

① 地址为 <http://css3pie.com/documentation/supported-css3-features>。

② 地址为 <http://weston.ruter.net/projects/css-gradients-via-canvas>。

渐变的方向、起始颜色、结束颜色。具有这三个参数就可以制作一个最简单、最普通的渐变效果。如果制作一个复杂的多色渐变效果，就需要在同一个渐变方向增添多个色标。具备这些渐变参数（至少三个），各浏览器就会绘制与渐变线垂直的颜色条来填充整个容器。浏览器渲染出来的效果就类似于制作软件设计出来的渐变图像。

10.2.1 CSS3 线性渐变语法与参数

相对于其他的 CSS3 属性的语法而言线性渐变的语法要复杂。早期的语法在各浏览器内核下语法不尽相同，特别是在 Webkit 内核之下还分新旧两种版本。接下来先从各浏览器下的语法入手，开始介绍 CSS3 的线性渐变语法。

1. Webkit 引擎的 CSS3 线性渐变语法与属性参数

Webkit 是第一个支持 CSS3 渐变的浏览器引擎，不过其语法也比其他浏览器引擎复杂，还分为新旧两个版本。

1) Webkit 引擎老式语法。

```
-webkit-gradient(<type>,<point>[,<radius>]?,<point>[,<radius>]?[,<stop>]*)
```

2) Webkit 引擎新式语法。

```
-webkit-linear-gradient([<point>||<angle>,<stop>,<stop>[,<stop>]*)
```

3) Webkit 引擎渐变属性参数。

`-webkit-gradient` 是 webkit 引擎对渐变的实现一共有五个参数。第一个参数 `type` 表示渐变类型，可以是线性渐变 `linear` 或者径向渐变 `radial`。第二个参数和第三个参数，都是一对值，分别表示渐变的起点位置和终点位置。这对值可以用坐标形式表示，也可以用关键值表示，比如“`left top`”（左上角）和“`left bottom`”（左下角）。第四个和第五个参数，分别是两个 `color-stop` 函数（色标），该函数接受两个参数，第一个表示渐变的位置，0 表示起点，0.5 为中点，1 为终点；第二个表示该点的颜色，如图 10-2 所示。

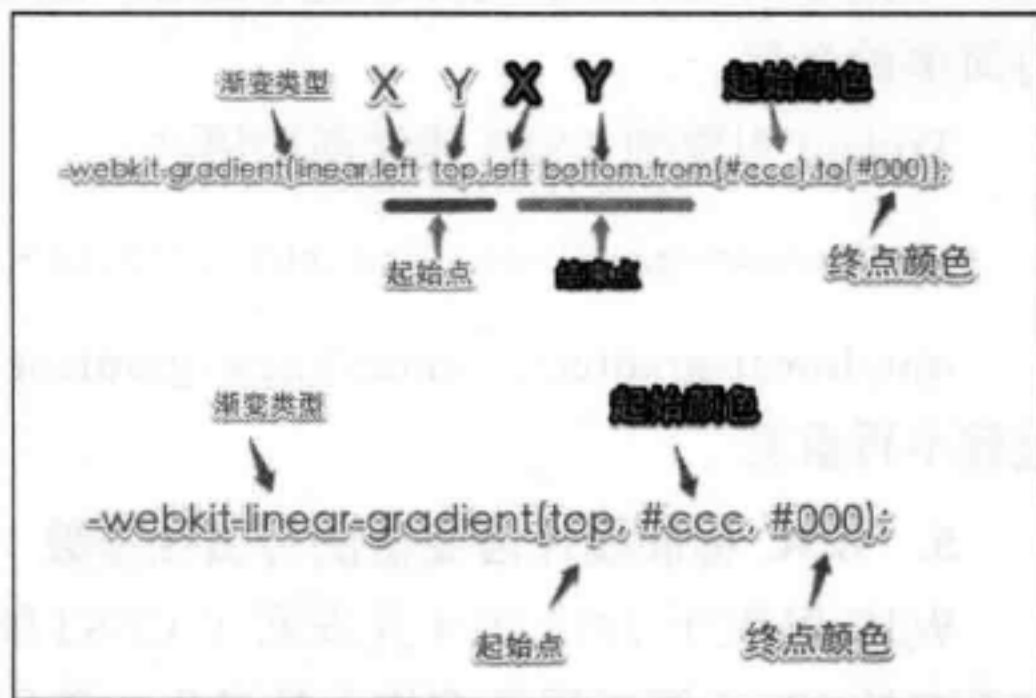


图 10-2 webkit 引擎的线性渐变语法

2. Gecko 引擎的 CSS3 的线性渐变语法与属性参数

Gecko 引擎的浏览器 Firefox 在 3.6 版本开始支持 CSS3 的线性渐变属性。Gecko 引擎与 Webkit 引擎的新版本渐变设计时用法基本相同，只是使用的私有前缀不同。

Gecko 引擎的渐变语法。

```
-moz-linear-gradient([<point>||<angle>,<stop>,<stop>[,<stop>]*)
```


在 Gecko 引擎的渐变中共有三个参数，第一个数表示线性渐变的方向，例如 top 是从上到下、left 是从左到右。如果定义成“left top”，就是从左上角到右下角。第二个和第三个参数分别是起点颜色和终点颜色。还可以在它们之间插入更多的参数，表示多种颜色的渐变，如图 10-3 所示。

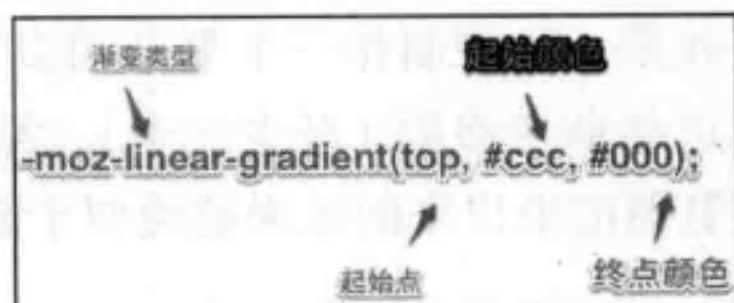


图 10-3 Gecko 引擎的线性渐变语法

3. Presto 引擎的 CSS3 线性渐变语法与属性参数

Presto 引擎的 Opera 浏览器在 11.6 版本开始支持 CSS3 的线性渐变。在 Presto 引擎浏览器中 CSS3 线性渐变的使用语法和 Gecko 引擎浏览器中的线性渐变的语法非常类似，唯一不同的是在 Presto 引擎浏览器中需要使用其自己的私有前缀为“-o-”。

Presto 引擎的线性渐变语法。

```
-o-linear-gradient([<point>||<angle>,<stop>,<stop>[,<stop>]*)
```

其中具有三个参数：第一个参数表示线性渐变的方向，top 表示从上到下，left 表示从左到右，如果定义成“left top”表示从左上角到右下角；第二个和第三个参数分别是起点颜色和结束颜色，还可以在它们之间插入更多的参数，表示多种颜色的渐变，如图 10-4 所示。

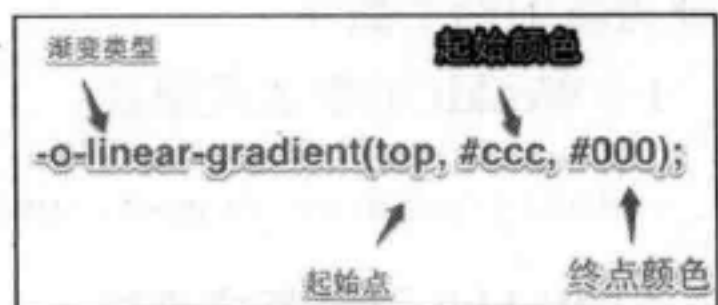


图 10-4 Presto 引擎下线性渐变语法

4. Trident 引擎的 CSS3 线性渐变语法与属性参数

Trident 引擎的浏览器主要有 IE，早期版本的 IE 浏览器是不支持 CSS3 线性渐变的属性，不过在 IE 10 开始支持了这个属性。这里主要针对 IE 10 + 浏览器的 CSS3 线性渐变进行简单的介绍。

Trident 引擎的 CSS3 线性渐变语法。

```
-ms-linear-gradient([<point>||<angle>,<stop>,<stop>[,<stop>]*)
```

-ms-linear-gradient、-moz-linear-gradient 以及 -o-linear-gradient 属性参数是一样的，这里就不再重复。

5. W3C 标准线性渐变语法与属性参数

W3C 组织于 2012 年 4 月发布了 CSS3 线性渐变的 CR 版本（候选人推荐版本）。这一次发布的 CSS3 渐变属性有很大的变化，使用语法较前面的版本要简单。最让大家感到高兴的是，到写本文的时候，所有现代浏览器都支持 W3C 的标准语法，包括 IE 浏览器，也在 IE 10 中支持了标准语法。

W3C 标准线性渐变语法如下。

```
linear-gradient([<angle> | to <side-or-corner> ],<color-stop>[,<color-stop>]+)
```

包括三个主要属性参数：第一个参数指定渐变的方向，同时决定渐变颜色的停止位置。这个参数值可以省略，当省略时其取值为 to bottom。在决定渐变的方向主要有两种方法。

□ **<angle>**：通过角度来确定渐变的方向。0 度表示渐变方向从下向上，90 度表示渐变方向从左向右。如果取值为负值，其角度按顺时针方向旋转。

□ **关键词**：通过关键词来确定渐变的方向。比如“to top”、“to right”、“to bottom”和“to left”。这些关键词对应的角度值为“0deg”、“90deg”、“180deg”和“270deg”。除了使用“to top”、“to left”之外，还可以使用“top left”左上角到右下角、“top right”左上角到右下角等。

第二个和第三个参数，表示颜色的起始点和结束点。大家可以从中插入更多的颜色值。

10.2.2 CSS3 线性渐变的基本用法

前几节中介绍了各引擎浏览器以及 W3C 标准中 CSS3 线性渐变的语法以及相关属性的基本知识。CSS3 在各浏览器下得到较好的支持，接下来以标准语法的使用，通过案例向大家展示 CSS3 线性渐变的基本使用。

1. 颜色从顶部向底部渐变

制作从顶部到底部直线渐变最简单的方法直接使用“to top”关键词，表示第一颜色向第二颜色渐变。实现类似于“to top”效果还可以使用角度值“0deg”、“360deg”和“-360deg”。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>CSS3 线性渐变——从顶部到底部渐变 </title>
  <style type="text/css" media="screen">
    div {
      width: 400px;
      height: 150px;
      border: 1px solid #666;
      line-height: 150px;
      text-align: center;
      font-weight: 900;
      font-size: 30px;
      color: #fff;
      margin: 10px auto;
    }
    .toTop {
      background-image: -webkit-linear-gradient(to top, orange, green);
      background-image: linear-gradient(to top, orange, green);
    }
    .toTop-deg {
      background-image: -webkit-linear-gradient(0deg, orange, green);
      background-image: linear-gradient(0deg, orange, green);
    }
    .toTop-deg2 {
      background-image: -webkit-linear-gradient(360deg, orange, green);
```

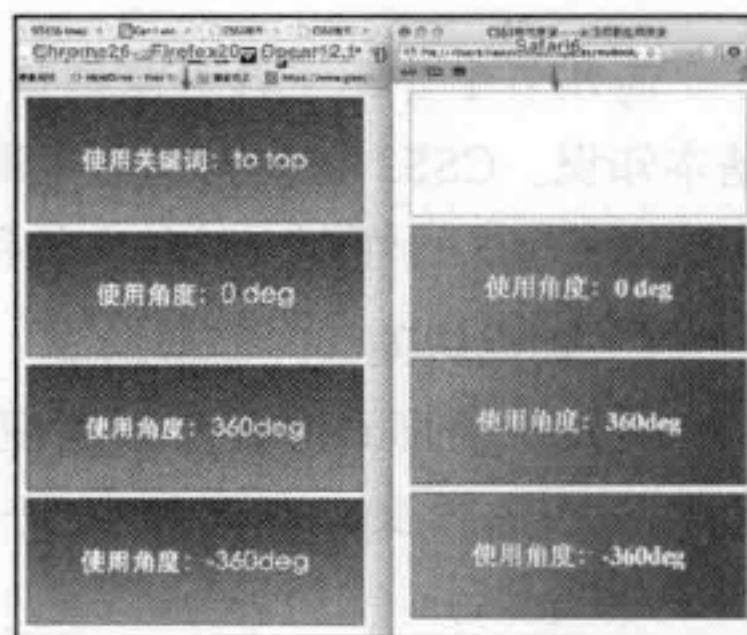
```

background-image:linear-gradient(360deg,orange,green);
}
.toTop-deg3 {
background-image:-webkit-linear-gradient(-360deg, orange, green);
background-image:linear-gradient(-360deg,orange,green);
}
</style>
</head>
<body>
<div class="toTop">使用关键词: top</div>
<div class="toTop-deg">使用角度: 0 deg</div>
<div class="toTop-deg2">使用角度: 360deg</div>
<div class="toTop-deg3">使用角度: -360deg</div>
</body>
</html>

```

效果如图 10-5 所示。

如图 10-5 所示, Safari 浏览器下还不支持 “to top” 关键词, 而且在角度值的解析渐变也不同。“to top” 所展示的是第一色从底部向顶部的第二色渐变, 如上例所示, 从 orange 向 green 渐变, 而且是从底部向顶部直线渐变。



☆图 10-5 实现顶部到底部渐变效果

2. 颜色从底部向顶部渐变

关键词 “to bottom” 刚好与 “to top” 相反, 从底部向顶部实现渐变效果。也可以使用角度值 “180deg” 和 “-180deg” 实现同等效果。

```

.toBottom {
background-image:-webkit-linear-gradient(to bottom, orange, green);
background-image:linear-gradient(to bottom,orange,green);
}
.toBottom-deg{
background-image:-webkit-linear-gradient(180deg, orange, green);
background-image:linear-gradient(180deg,orange,green);
}
.toBottom-deg2{
background-image:-webkit-linear-gradient(-180deg, orange, green);
background-image:linear-gradient(-180deg,orange,green);
}

```

其效果如图 10-6 所示。

“to bottom” 实现了第一色从顶部向底部的第二色直线渐变, orange 向 green 渐变, 而且是从顶部向底部渐变。

3. 颜色从左向右渐变

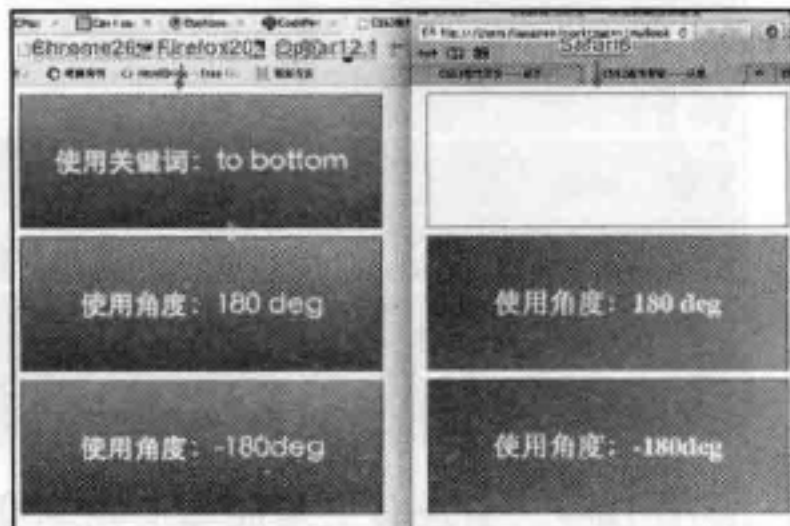
“to left” 关键词实现了从左向右颜色渐变, 也可以通过角度值 “90deg” 和 “270deg” 实现。

```

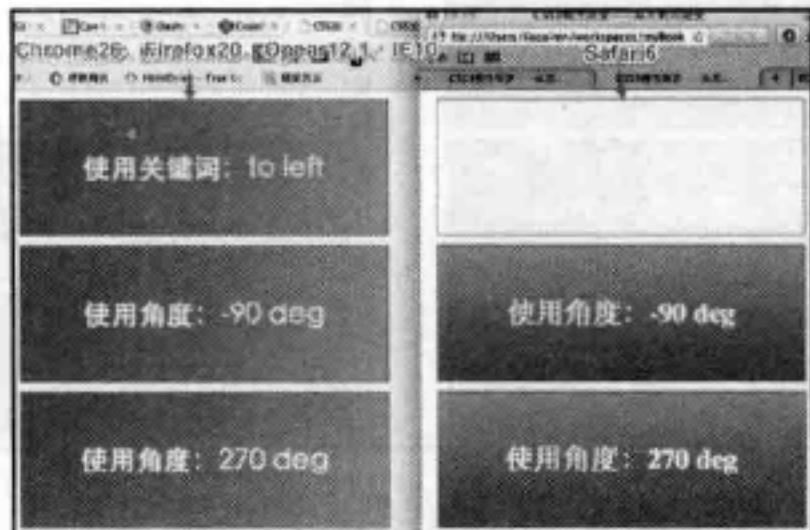
.toLeft {
    background-image:-webkit-linear-gradient(to left, orange, green);
    background-image:linear-gradient(to left,orange,green);
}
.toLeft-deg{
    background-image:-webkit-linear-gradient(-90deg, orange, green);
    background-image:linear-gradient(-90deg,orange,green);
}
.toLeft-deg2{
    background-image:-webkit-linear-gradient(270deg, orange, green);
    background-image:linear-gradient(270deg,orange,green);
}

```

效果如图 10-7 所示。



☆图 10-6 实现底部到顶部的直线渐变效果



☆图 10-7 实现左向右渐变效果

4. 颜色从右向左渐变

“to right”正好与“to left”效果相反。实现了颜色从右向左直线渐变，也可以使用角度值“90deg”和“-270deg”。

```

.toRight {
    background-image:-webkit-linear-gradient(to right, orange, green);
    background-image:linear-gradient(to right,orange,green);
}
.toRight-deg{
    background-image:-webkit-linear-gradient(90deg, orange, green);
    background-image:linear-gradient(90deg,orange,green);
}
.toRight-deg2{
    background-image:-webkit-linear-gradient(-270deg, orange, green);
    background-image:linear-gradient(-270deg,orange,green);
}

```

效果如图 10-8 所示。

5. 从右下角向左上角线性渐变

“to top left”实现从右下角向左上角线性渐变。

```

.toTopLeft {
    background-image:-webkit-linear-gradient(to top left, orange, green);

```

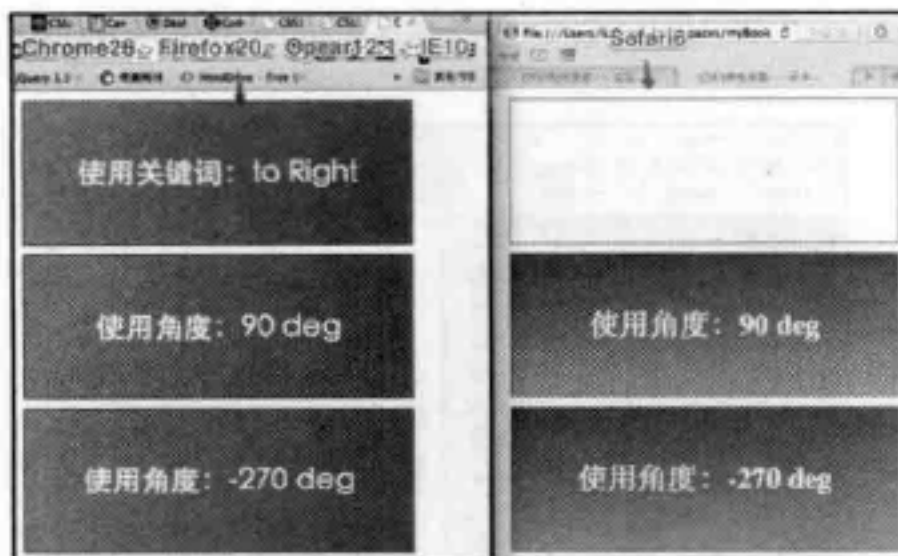


```

background-image:linear-gradient(to top left,orange,green);
}
.toTopLeft-deg{
background-image:-webkit-linear-gradient(315deg, orange, green);
background-image:linear-gradient(315deg,orange,green);
}
.toTopLeft-deg2{
background-image:-webkit-linear-gradient(-45deg, orange, green);
background-image:linear-gradient(-45deg,orange,green);
}

```

效果如图 10-9 所示。其效果与角度值“315 deg”和“-45deg”类似，但细节上略有不同。



☆图 10-8 实现右向左线性渐变效果



☆图 10-9 实现右下角向左上角渐变效果

6. 从左下角到右上角线性渐变

“to top right” 关键词实现左下角到右上角的线性渐变。

```

.toTopRight {
background-image:-webkit-linear-gradient(to top right, orange, green);
background-image:linear-gradient(to top right,orange,green);
}
.toTopRight-deg{
background-image:-webkit-linear-gradient(45deg, orange, green);
background-image:linear-gradient(45deg,orange,green);
}
.toTopRight-deg2{
background-image:-webkit-linear-gradient(-315deg, orange, green);
background-image:linear-gradient(-315deg,orange,green);
}

```

效果如图 10-10 所示。其效果与角度值“315deg”和“-45deg”效果类似，但细节上略有不同。

7. 从右上角到左下角线性渐变

“to bottom left” 关键词实现了右上角向左下角直线渐变。

```

.toBottomLeft {
background-image:-webkit-linear-gradient(to bottom left, orange, green);
background-image:linear-gradient(to bottom left,orange,green);
}

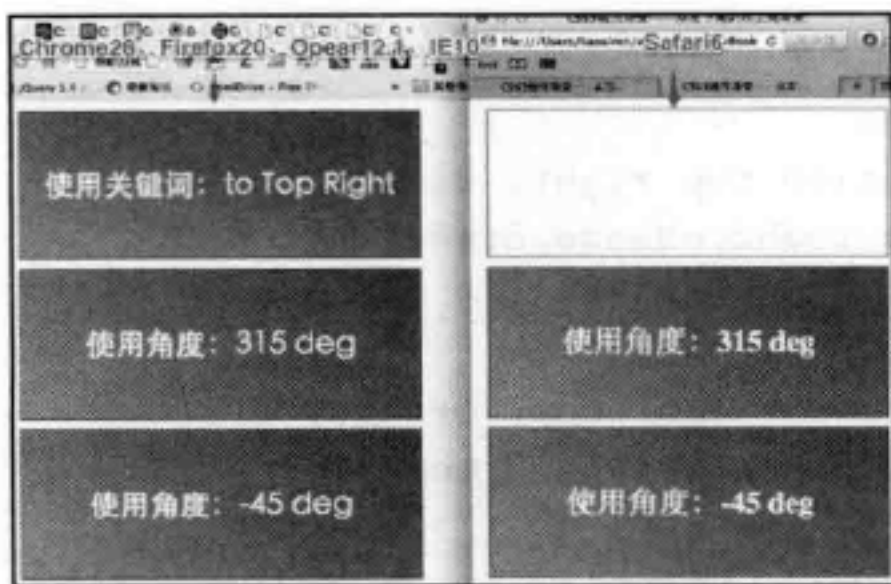
```

```

.toBottomLeft-deg{
    background-image:-webkit-linear-gradient(225deg, orange, green);
    background-image:linear-gradient(225deg,orange,green);
}
.toBottomLeft-deg2{
    background-image:-webkit-linear-gradient(-135deg, orange, green);
    background-image:linear-gradient(-135deg,orange,green);
}

```

效果如图 10-11 所示。其效果与角度值“225deg”和“-135deg”效果类似，但细节上略有不同。



☆图 10-10 实现左下角向右上角线性渐变



☆图 10-11 实现右上角向左下角线性渐变

8. 从左上角向右下角线性渐变

to bottom right 关键词实现了左上角向右下角直线渐变。

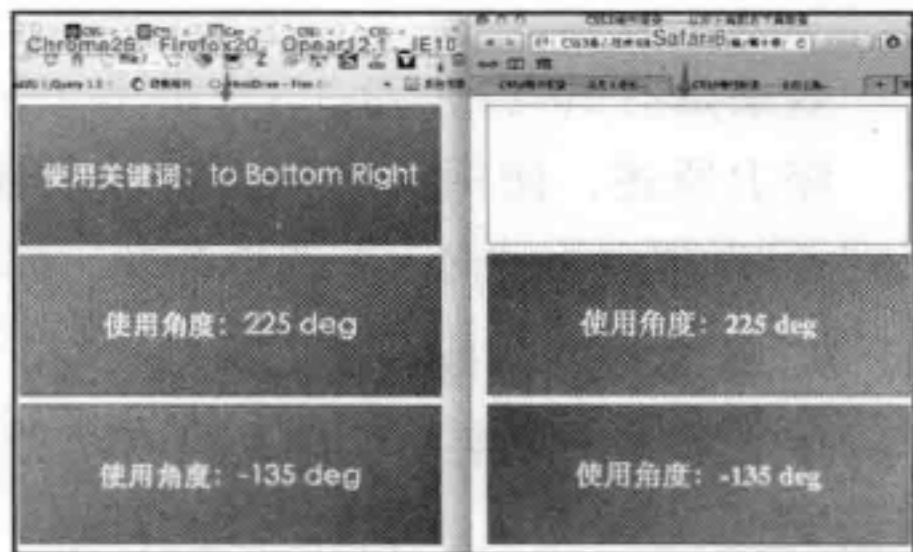
```

.toBottomRight {
    background-image:-webkit-linear-gradient(to bottom right, orange, green);
    background-image:linear-gradient(to bottom right,orange,green);
}
.toBottomRight-deg{
    background-image:-webkit-linear-gradient(135deg, orange, green);
    background-image:linear-gradient(135deg,orange,green);
}
.toBottomRight-deg2{
    background-image:-webkit-linear-gradient(-225deg, orange, green);
    background-image:linear-gradient(-225deg,orange,green);
}

```

效果如图 10-12 所示。其效果与角度值“225deg”和“-135deg”实现的效果类似，但在细节上略有不同。

上面主要演示了关键词与对应角度值实现的渐变效果。其中角度渐变取值多项，如“to top left”实现了右下角向左上角渐变，而与其实现同等效果的除了对应的角度值 315deg 和 -45deg



☆图 10-12 实现左上角到右下角线性渐变

之外，与“to left top”关键词实现的效果是一样的。换句话说，“to top left”和“top left top”关键词实现的效果是相同的。按同样的原理，其他角度值也有相对应的关键词。

```

/* 向左上角渐变 */
.toTopLeft {
    background-image:-webkit-linear-gradient(to top left, orange, green);
    background-image:linear-gradient(to top left,orange,green);
}
.toLeftTop {
    background-image:-webkit-linear-gradient(to left top, orange, green);
    background-image:linear-gradient(to left top,orange,green);
}
/* 向右上角渐变 */
.toTopRight{
    background-image:-webkit-linear-gradient(to top right, orange, green);
    background-image:linear-gradient(to top right,orange,green);
}
.toRightTop {
    background-image:-webkit-linear-gradient(to right top, orange, green);
    background-image:linear-gradient(to right top,orange,green);
}
/* 向左下角渐变 */
.toBottomLeft {
    background-image:-webkit-linear-gradient(to bottom left, orange, green);
    background-image:linear-gradient(to bottom left,orange,green);
}
.toLeftBottom {
    background-image:-webkit-linear-gradient(to left bottom, orange, green);
    background-image:linear-gradient(to left bottom,orange,green);
}
/* 向右下角渐变 */
.toBottomRight {
    background-image:-webkit-linear-gradient(to bottom right, orange, green);
    background-image:linear-gradient(to bottom right,orange,green);
}
.toRightBottom {
    background-image:-webkit-linear-gradient(to right bottom, orange, green);
    background-image:linear-gradient(to right bottom,orange,green);
}

```

效果如图 10-13 所示。

综上所述，使用关键词实现的线性渐变效果可以从关键词的方向性来定，例如“to top”可以理解为“向上”线性渐变。简单理解就是第一颜色从下向上到第二颜色线性渐变。

9. 多色线性渐变

在实际中，渐变不仅仅是只有两种颜色，会有多色。接下来，一起来看从左向右的五彩渐变。

```

.toLeft {

```



```

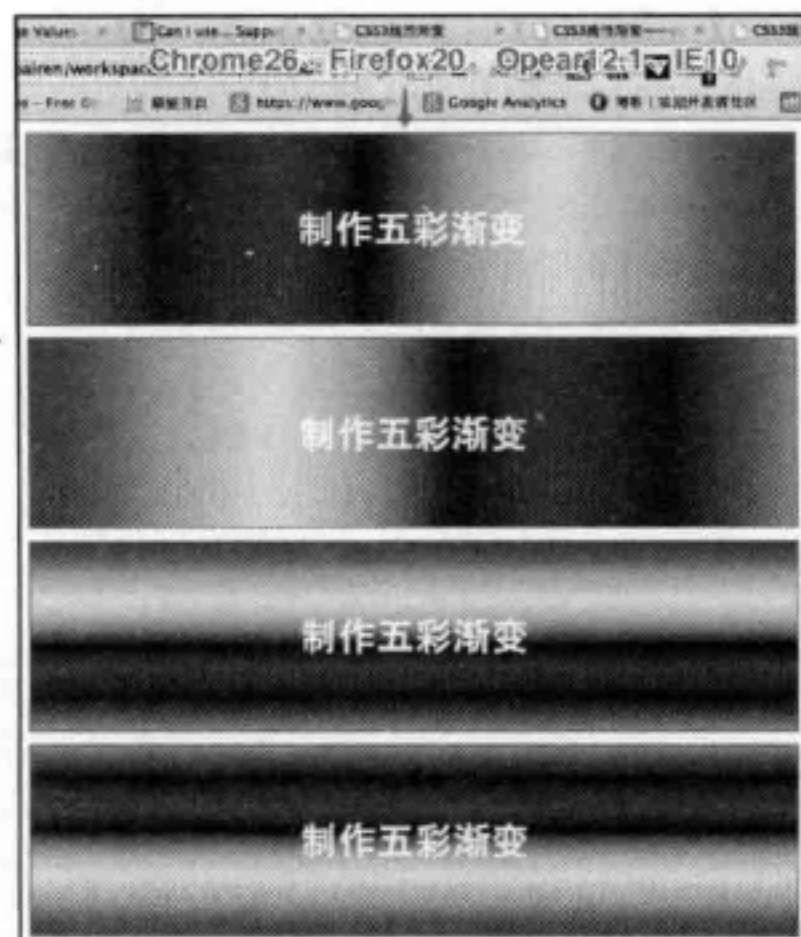
background-image:-webkit-linear-gradient(to left, red,
orange,yellow,green,blue,indigo,violet);
background-image:linear-gradient(to left, red, orange,
yellow,green,blue,indigo,violet);
}
.toRight {
background-image:-webkit-linear-gradient(to right, red,
orange,yellow,green,blue,indigo,violet);
background-image:linear-gradient(to right, red,
orange,yellow,green,blue,indigo,violet);
}
.toTop {
background-image:-webkit-linear-gradient(to top, red,
orange,yellow,green,blue,indigo,violet);
background-image:linear-gradient(to top, red, orange,
yellow,green,blue,indigo,violet);
}
.toBottom {
background-image:-webkit-linear-gradient(to bottom, red,
orange,yellow,green,blue,indigo,violet);
background-image:linear-gradient(to bottom, red,
orange,yellow,green,blue,indigo,violet);
}

```

效果如图 10-14 所示。



☆图 10-13 对应关键词实现的线性渐变效果



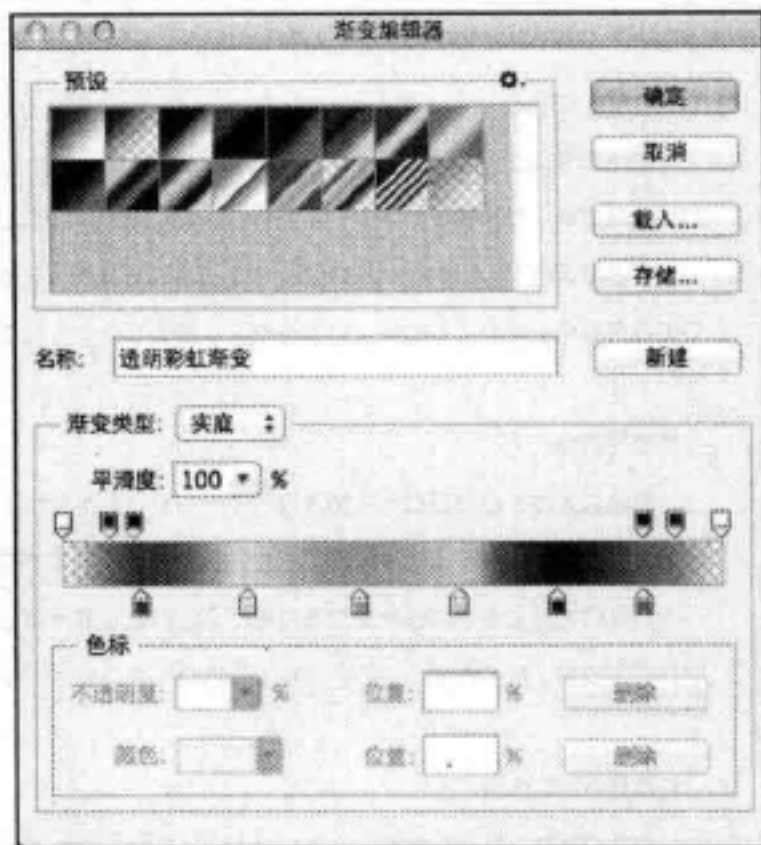
☆图 10-14 制作五彩渐变

10.2.3 自定义 CSS3 线性渐变

现在我们理解了如何声明线性渐变，下面来声明自己的渐变。

在实现 PSD 转换 Web 页面时，有时候在页面的设计图中会包含渐变效果。为了实现设计效果图中的渐变效果一样，可以借助相关的制作软件，例如 Photoshop 制图软件。通过制作软件打开设计原文件，并通过制图工具中渐变工具确定相关的渐变参数，如图 10-15 所示。

通过 Photoshop 屏幕截图可以注意到，颜色从 0% 的不透明度开始，第一个色标的位置位于 0%，其透明度为 0%，第二个色标位置为 80% 的不透明度，位置位于 8%。可以使用这个渐变工具从 CSS 声明中捕捉相关数据，实现自定义线性渐变。



☆图 10-15 Photoshop 中的示例线性渐变

```
.toLeft {
  background-image:-webkit-linear-gradient(
    to left,
    rgba(255,0,0,0) 0%,
    rgba(255,0,0,0.8) 7%,
    rgba(255,0,0,1) 11%,
    rgba(255,0,0,1) 12%,
    rgba(255,252,0,1) 28%,
    rgba(1,180,7,1) 45%,
    rgba(0,234,255,1) 60%,
    rgba(0,3,144,1) 75%,
    rgba(255,0,198,1) 88%,
    rgba(255,0,198,0.8) 93%,
    rgba(255,0,198,0) 100%);
  background-image:linear-gradient(
    to left,
    rgba(255,0,0,0) 0%,
    rgba(255,0,0,0.8) 7%,
    rgba(255,0,0,1) 11%,
    rgba(255,0,0,1) 12%,
    rgba(255,252,0,1) 28%,
    rgba(1,180,7,1) 45%,
    rgba(0,234,255,1) 60%,
    rgba(0,3,144,1) 75%,
    rgba(255,0,198,1) 88%,
    rgba(255,0,198,0.8) 93%,
    rgba(255,0,198,0) 100%);
}

.toRight {
  background-image:-webkit-linear-gradient(
```

```

to right,
rgba(255,0,0,0) 0%,
rgba(255,0,0,0.8) 7%,
rgba(255,0,0,1) 11%,
rgba(255,0,0,1) 12%,
rgba(255,252,0,1) 28%,
rgba(1,180,7,1) 45%,
rgba(0,234,255,1) 60%,
rgba(0,3,144,1) 75%,
rgba(255,0,198,1) 88%,
rgba(255,0,198,0.8) 93%,
rgba(255,0,198,0) 100%);
background-image: linear-gradient(
to right,
rgba(255,0,0,0) 0%,
rgba(255,0,0,0.8) 7%,
rgba(255,0,0,1) 11%,
rgba(255,0,0,1) 12%,
rgba(255,252,0,1) 28%,
rgba(1,180,7,1) 45%,
rgba(0,234,255,1) 60%,
rgba(0,3,144,1) 75%,
rgba(255,0,198,1) 88%,
rgba(255,0,198,0.8) 93%,
rgba(255,0,198,0) 100%);
}

```

效果如图 10-16 所示。



☆图 10-16 自定义直线渐变

上图再次证明，一个渐变可以包含多个色标，位置值为 0 ~ 1 之间的小数，或者 0% ~ 100% 之间的百分数，指定色标的位置比例，其用法与 Photoshop 中的值线渐变工具用法相似，如图 10-15 所示。

10.2.4 实战体验：CSS3 制作渐变按钮

Web 页面或者说 Web 程序应用中，按钮是设计中的一个重要元素，也是设计师非常重视的一个设计元素。设计师借助 Photoshop 等软件实现按钮效果，然后通过图片的方式转用到 Web 页面中或者 Web 应用程序中。这种方法安全、兼容性较强，实现方法也简单，但适应能力比较弱、重用性、可扩展性、维护性差。例如，修改一个按钮的颜色，必须先从制作软件中修改好，再次切图，最后替换 Web 页面中的图片。

CSS3 的渐变出现后，Web 设计师可以摆脱 Photoshop 的束缚，特别是在配合 CSS3 的阴影、圆角和 @font-face（第 15 章介绍）等属性，可以直接脱离制图软件，直接使用 CSS 快速设计各种精巧、靓丽、细腻的按钮（几乎可以和设计软件制作出来的效果一模一样）。

通过 CSS3 属性制作的按钮与设计软件制作的按钮相比，有以下几大优势。

□ 灵活性、可扩展性强：可以通过 CSS3 的属性可以直接修改按钮的大小、背景等效果。

- ❑ 可以减少 http 请求, 提高页面的加载性能。
- ❑ 可以应用到任何的 HTML 标签元素, 如 a、input、button、span 和 div 等。
- ❑ 可以支持按钮多种状态效果, 比如当前状态、悬停状态和点击状态。
- ❑ 完全兼容不支持 CSS3 的浏览器, 如果不兼容 CSS3, 则显示没有渐变和阴影的普通效果。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>CSS3 线性渐变制作按钮 </title>
  <style type="text/css" media="screen">
/* 加载 google web 字体 */
@import url(http://fonts.googleapis.com/css?family=Arvo);
body {
  background: #f8fcd4;
  /* 制作背景渐变效果 */
  background: -webkit-linear-gradient(to bottom, #f8fcd4 19%, #dbfacb 100%);
  background: linear-gradient(to bottom, #f8fcd4 19%, #dbfacb 100%);
  background-attachment: fixed;
}
body > div {
  text-align: center;
  display: block;
  width: 800px;
  margin: 50px auto;
}
/* 加载本地字体制作图标 */
@font-face {
  font-family: 'EntypoRegular';
  src: url('font/entypo-webfont.eot');
  src: url('font/entypo-webfont.eot?#iefix') format('embedded-opentype'),
    url('font/entypo-webfont.woff') format('woff'),
    url('font/entypo-webfont.ttf') format('truetype'),
    url('font/entypo-webfont.svg#EntypoRegular') format('svg');
  font-weight: normal;
  font-style: normal;
}
/* 设置按钮基本样式 */
.button,
.button-bevel {
  font-family: Arial, Helvetica, sans-serif;
  font-size: 13px;
  color: #fff;
  text-decoration: none;
  display: inline-block;
  text-align: center;
  padding: 7px 20px 9px;
  margin: .5em .5em .5em 0;
```

```

    cursor: pointer;
    /* 设置按钮文本阴影效果 */
    text-shadow: 0 1px 1px rgba(0,0,0,0.4);
    text-transform: capitalize;
    /* 设置按钮过渡动画效果 */
    transition: 0.1s linear;
}
.button {
    border-radius: 2px; /* 设置圆角效果 */
    box-shadow: inset rgba(255,255,255,0.3) 1px 1px 0; /* 设置盒子阴影 */
}
/* 设置按钮悬停效果基本样式 */
.button:hover,
.button-bevel:hover {
    color: #fff;
    text-decoration: none;
}
/* 设置按钮点击效果基本样式 */
.button:active {
    box-shadow: inset rgba(0,0,0,0.4) 0px 0px 6px; /* 点击时修改盒子阴影效果 */
}
/* 设置圆角按钮样式 */
.rounded {
    border-radius: 20px;
}
/* 制作橙色按钮 */
.orange {
    background: rgb(255,183,0);
    /* 制作渐变效果 (向下渐变) */
    background: -webkit-linear-gradient(to bottom, rgba(255,183,0,1) 0%,
        rgba(255,140,0,1) 100%);
    background: linear-gradient(to bottom, rgba(255,183,0,1) 0%,
        rgba(255,140,0,1) 100%);
    border: 1px solid #e59500;
}
/* 制作橙色按钮悬停效果 */
.orange:hover {
    background: rgb(255,203,72);
    /* 制作渐变效果 (向下渐变) */
    background: -webkit-linear-gradient(to bottom, rgba(255,203,72,1) 0%,
        rgba(255,156,35,1) 100%);
    background: linear-gradient(to bottom, rgba(255,203,72,1) 0%,
        rgba(255,156,35,1) 100%);
}
/* 其他颜色参考随书代码 */
/* 制作洋红色按钮 */
/* 制作青色按钮 */
/* 制作大红按钮 */
/* 制作黑色按钮 */
/* 制作绿色按钮 */

```

```

/* 制作 3D 立体按钮基本样式 */
.button-bevel {
    vertical-align: top;
    border-radius: 4px; /* 设置圆角效果 */
    border: none;
    padding: 10px 25px 12px;
}
/* 3D 按钮点击状态效果 */
.button-bevel:active {
    position: relative;
    top: 5px;
}
/* 制作橙色 3D 按钮立体效果 */
.button-bevel.orange {
    box-shadow: #c46d00 0 6px 0px, rgba(0, 0, 0, 0.3) 0 10px 3px;
}
/* 制作橙色 3D 按钮点击状态效果 */
.button-bevel.orange:active {
    box-shadow: #c46d00 0 3px 0, rgba(0, 0, 0, 0.2) 0 6px 3px;
}
/* 其他颜色参考随书代码 */
/* 制作洋红色 3D 按钮立体效果 */
/* 制作青色 3D 立体按钮效果 */
/* 制作大红 3D 立体按钮效果 */
/* 制作黑色 3D 立体按钮效果 */
/* 制作绿色 3D 立体按钮效果 */
/* 设置按钮图标基本样式 */
.button span,
.button-bevel span {
    font-family: 'EntypoRegular';
    font-size: 20px;
    font-weight: normal;
    vertical-align: middle;
    line-height: 0;
    margin-right: .1em;
}
/* 使用 @font-face 制作图标 */
.refresh:after { content: "h"; font-size: 34px; }
.shuffle:after { content: "f"; font-size: 34px; }
.preview:after { content: "M"; font-size: 34px; }
.tea:after { content: "u"; font-size: 34px; }
.wifi:after { content: "T"; font-size: 34px; }
.locator:after { content: "O"; font-size: 34px; }

.rss:after { content: "S"; font-size: 34px; }
.cloud:after { content: "y"; font-size: 34px; }
.download:after { content: "w"; font-size: 34px; }
.trash:after { content: "I"; font-size: 34px; }
.rack:after { content: "t"; font-size: 34px; }
.map:after { content: "l"; font-size: 34px; }

```



```

.setting:after { content: "@"; font-size: 34px; }
.identity:after { content: "."; font-size: 34px; }
.navigation:after { content: "2"; font-size: 34px; }
.gallery:after { content: "p"; font-size: 34px; }
.email:after { content: "%"; font-size: 34px; }
.users:after { content: "+"; font-size: 34px; }

.calendar:after { content: "P"; font-size: 34px; }
.link:after { content: ">"; font-size: 34px; }
.time:after { content: "N"; font-size: 34px; }
.folder:after { content: "s"; font-size: 34px; }
.tag:after { content: "C"; font-size: 34px; }
.share:after { content: "5"; font-size: 34px; }

.lock:after { content: "U"; font-size: 34px; }
.unlock:after { content: "V"; font-size: 34px; }
.mic:after { content: "O"; font-size: 34px; }
.love:after { content: "6"; font-size: 34px; }
.star:after { content: "7"; font-size: 34px; }
.like:after { content: "8"; font-size: 34px; }

.phone:after { content: "!"; font-size: 34px; }
.flag:after { content: "?"; font-size: 34px; }
.adduser:after { content: "-"; font-size: 34px; }
.attach:after { content: "'"; font-size: 34px; }
.comment:after { content: ":"; font-size: 34px; }
.edit:after { content: "&"; font-size: 34px; }

.upload:after { content: "v"; font-size: 34px; }
.storage:after { content: "x"; font-size: 34px; }
.photo:after { content: "D"; font-size: 34px; }
.help:after { content: "K"; font-size: 34px; }
.glass:after { content: "R"; font-size: 34px; }
.print:after { content: "<"; font-size: 34px; }

.gadget:after { content: "'"; font-size: 34px; }

</style>
</head>
<body>
  <div data-for="beveled">
    <div>
      <a href="javascript:void(0)" class="button-bevel orange">
        <span class="refresh"></span> Refresh </a>
      <a href="javascript:void(0)" class="button-bevel orange">
        <span class="shuffle"></span> Shuffle </a>
      <a href="javascript:void(0)" class="button-bevel orange">
        <span class="preview"></span> Preview </a>
      <a href="javascript:void(0)" class="button-bevel orange">
        <span class="tea"></span> Tea </a>
    </div>
  </div>

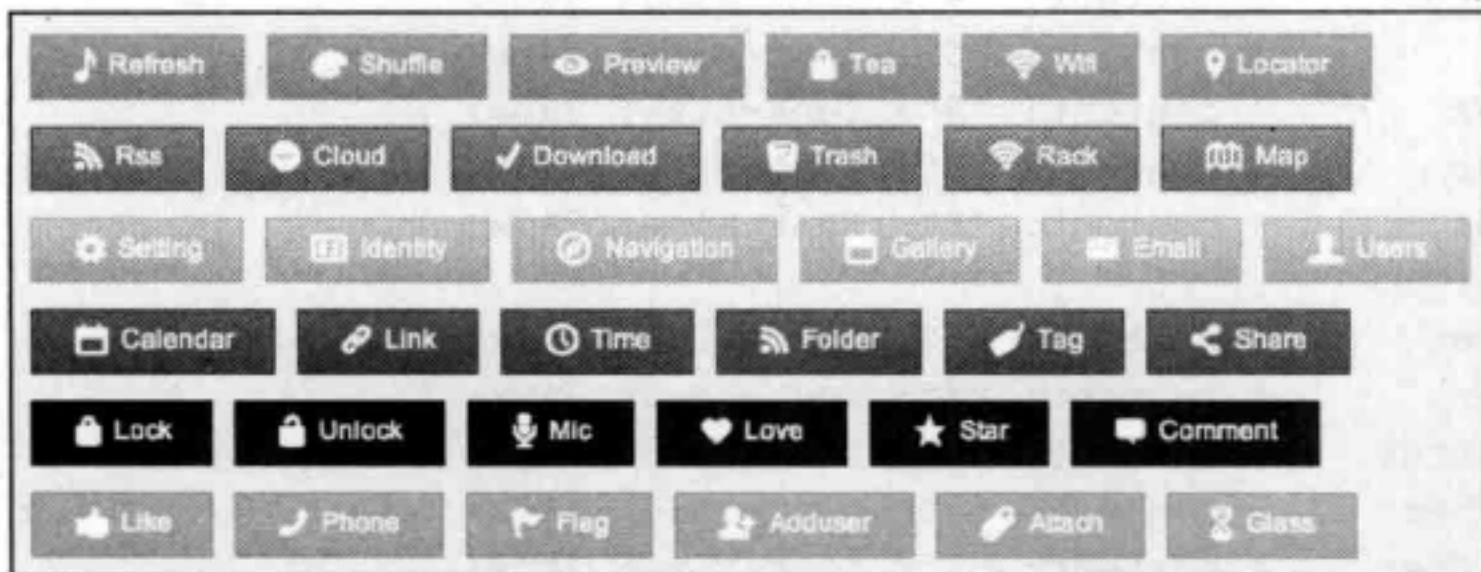
```

```

    <a href="javascript:void(0)" class="button-bevel orange">
    <span class="wifi"></span> Wifi </a>
    <a href="javascript:void(0)" class="button-bevel orange">
    <span class="locator"></span> Locator </a>
  </div>
</div>
/* 省略类似按钮结构 */
</div>
</body>
</html>

```

上例通过修改颜色样式类，可以设计不同色彩风格的按钮效果。例如示例中设置了 **orange** 类名实现橙色按钮、**magenta** 类名实现洋红色按钮、**cyan** 类名实现青色按钮、**red** 类名实现大红色按钮、**black** 类名实现黑色按钮和 **green** 类名实现绿色按钮，如图 10-17 所示。



☆图 10-17 CSS3 渐变 linear-gradient 配合 box-shadow 实现内阴影发光按钮效果

使用类名 **button-bevel** 配合 CSS3 的 **box-shadow** 属性制作 3D 立体效果，如下所示。

```

/* 制作 3D 立体按钮基本样式 */
.button-bevel {
  vertical-align: top;
  border-radius: 4px; /* 设置圆角效果 */
  border: none;
  padding: 10px 25px 12px;
}
/* 3D 按钮点击状态效果 */
.button-bevel:active {
  position: relative;
  top: 5px;
}
/* 制作橙色 3D 按钮立体效果 */
.button-bevel.orange {
  box-shadow: #c46d00 0 6px 0px, rgba(0, 0, 0, 0.3) 0 10px 3px;
}
/* 制作橙色 3D 按钮点击状态效果 */
.button-bevel.orange:active {
  box-shadow: #c46d00 0 3px 0, rgba(0, 0, 0, 0.2) 0 6px 3px;
}

```

```

/* 制作洋红色 3D 按钮立体效果 */
.button-bevel.magenta {
    box-shadow: #ca075c 0 6px 0px, rgba(0, 0, 0, 0.3) 0 10px 3px;
}
/* 制作洋红色 3D 按钮点击状态效果 */
.button-bevel.magenta:active {
    box-shadow: #ca075c 0 3px 0, rgba(0, 0, 0, 0.2) 0 6px 3px;
}
/* 制作青色 3D 立体按钮效果 */
.button-bevel.cyan {
    box-shadow: #1994d3 0 6px 0px, rgba(0, 0, 0, 0.3) 0 10px 3px;
}
/* 制作青色 3D 立体按钮点击效果 */
.button-bevel.cyan:active {
    box-shadow: #1994d3 0 3px 0, rgba(0, 0, 0, 0.2) 0 6px 3px;
}
/* 制作大红 3D 立体按钮效果 */
.button-bevel.red {
    box-shadow: #88180e 0 6px 0px, rgba(0, 0, 0, 0.3) 0 10px 3px;
}
/* 制作大红 3D 立体按钮点击效果 */
.button-bevel.red:active {
    box-shadow: #88180e 0 3px 0, rgba(0, 0, 0, 0.2) 0 6px 3px;
}
/* 制作黑色 3D 立体按钮效果 */
.button-bevel.black {
    box-shadow: #000 0 6px 0px, rgba(0, 0, 0, 0.3) 0 10px 3px,
    inset rgba(255, 255, 255, 0.3) 0 0 3px;
}
/* 制作黑色 3D 立体按钮点击效果 */
.button-bevel.black:active {
    box-shadow: #000 0 3px 0, rgba(0, 0, 0, 0.2) 0 6px 3px,
    inset rgba(255, 255, 255, 0.3) 0 0 3px;
}
/* 制作绿色 3D 立体按钮效果 */
.button-bevel.green {
    box-shadow: #439230 0 6px 0px, rgba(0, 0, 0, 0.3) 0 10px 3px;
}
/* 制作绿色 3D 立体按钮点击效果 */
.button-bevel.green:active {
    box-shadow: #439230 0 3px 0, rgba(0, 0, 0, 0.2) 0 6px 3px;
}

```

通过 box-shadow 多个阴影实现的 3D 立体效果，如图 10-18 所示。

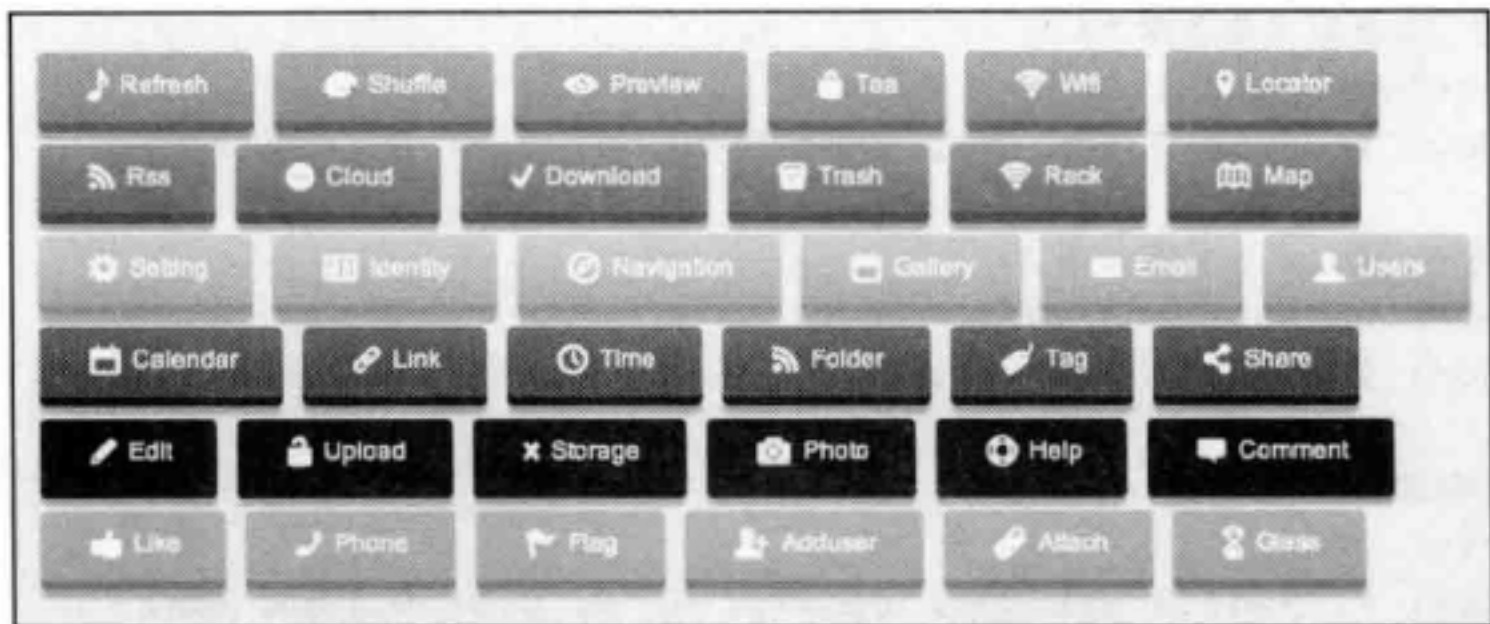
另外，给按钮添加类名 rounded 配合 border-radius 属性，轻松实现圆角按钮效果。

```

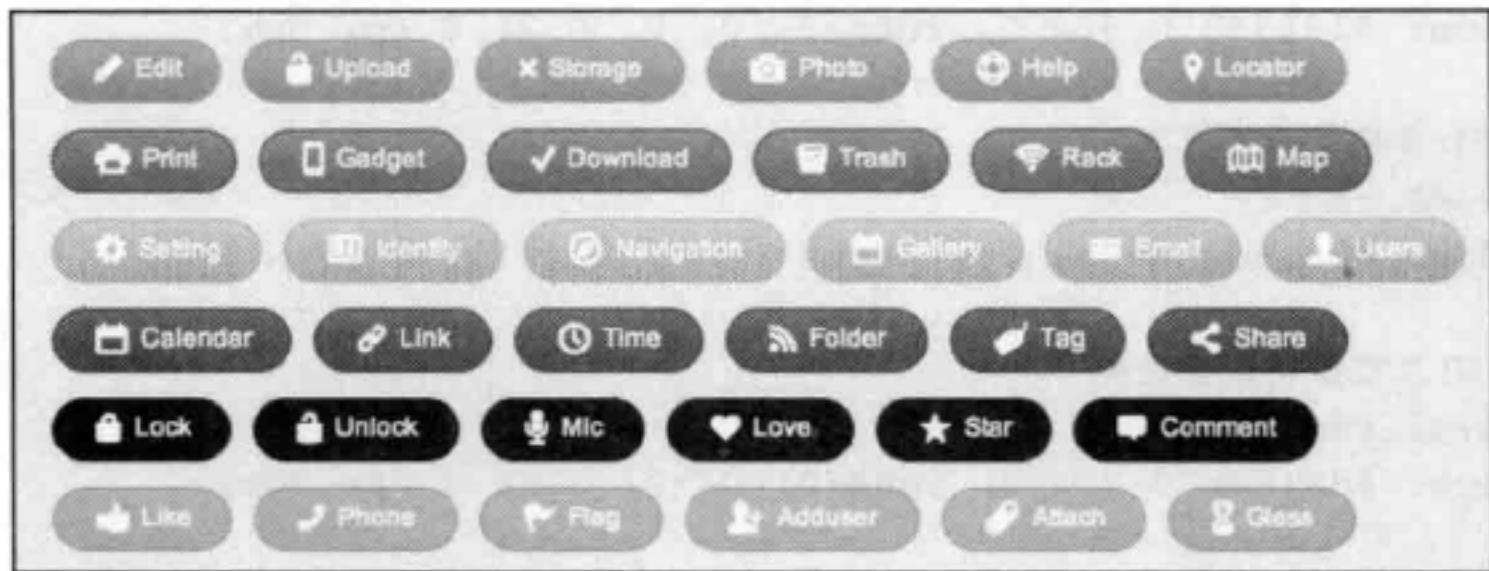
.rounded {
    border-radius: 20px;
}

```

效果如图 10-19 所示。

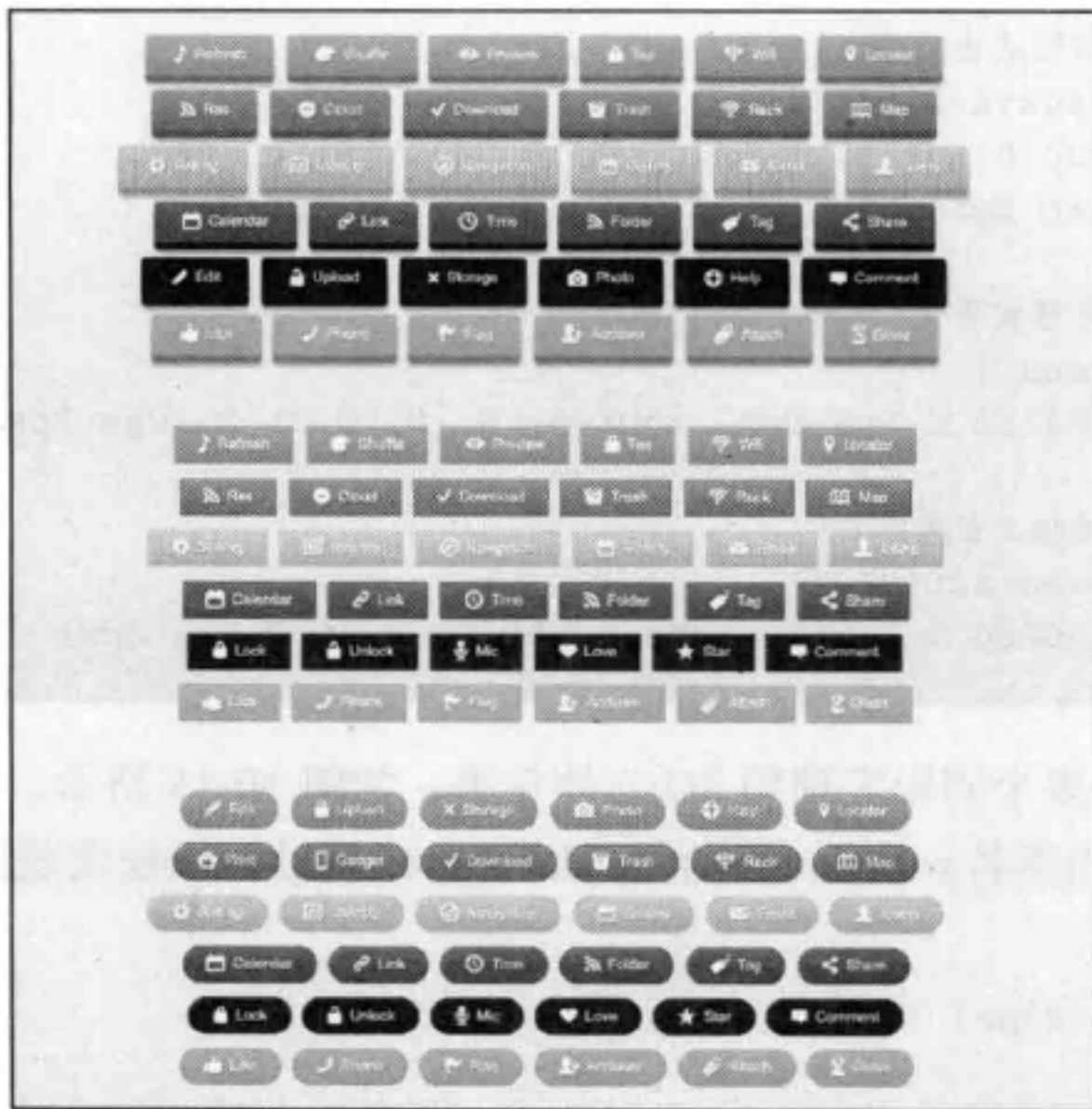


☆图 10-18 CSS3 渐变 linear-gradient 属性和 box-shadow 属性实现 3D 立体按钮效果



☆图 10-19 CSS3 渐变 linear-gradient 属性 border-radius 属性实现圆角按钮效果

整个示例的效果如图 10-20 所示。



☆图 10-20 CSS3 的 linear-gradient、box-shadow 和 border-radius 制作按钮效果

除了上面这个简单的实例之外,还可以使用 CSS3 的 `linear-gradient` 属性配合其他属性制作一些其他按钮或者其他效果。

10.3 CSS3 径向渐变

CSS3 径向渐变是圆形或椭圆形渐变。颜色不再沿着一条直线轴变化,而是从一个起点朝所有方向混合。但相对线性渐变要比径向渐变复杂得多。

10.3.1 CSS3 径向渐变语法

CSS3 的径向渐变已得到众多浏览器引擎的支持,只不过其语法的版本根据不同的引擎浏览器也不一样,特别是在 Webkit 引擎的浏览器下和线性渐变的语法类似,也有新旧之分。而在其他几大引擎的浏览器,其语法基本类似,只是使用的前缀不同而已。特别是在 2013 年 4 月, W3C 为 CSS3 径向渐变推出最新的语法格式。接下来一起来看看各引擎浏览器下的径向渐变语法。

1. Webkit 引擎的 CSS3 径向渐变语法

CSS3 径向渐变在 Webkit 引擎下的语法和线性渐变的语法一样,分为两种,一种是旧版本的语法,另外一种是新版本语法。

Webkit 引擎旧版本语法:

```
-webkit-gradient([<type>],[<position> || <angle>],[<shape> ||  
    <size>],[<color-stop>,<color-stop>[,<color-stop>]*]);
```

Webkit 引擎新版本语法:

```
-webkit-radial-gradient([<position> || <angle>],[<shape> ||  
    <size>],[<color-stop>,<color-stop>[,<color-stop>]*]);
```

Webkit 引擎中的浏览器 Chrome 4 ~ 9 和 Safari 4 ~ 5 版本支持老式的 Webkit 引擎径向语法, Chrome 10.0+ 和 Safari 5.1+ 支持 Webkit 引擎新式径向渐变。

2. Gecko 引擎的 CSS3 的径向渐变语法

Gecko 引擎的 CSS3 的径向渐变和 Webkit 引擎新式语法类似,只是使用的前缀不同而已。

```
-moz-radial-gradient([<position> || <angle>],[<shape> ||  
    <size>],[<color-stop>,<color-stop>[,<color-stop>]*]);
```

Gecko 引擎的 Firefox 浏览器中 Firefox 3.6+ 版本支持径向渐变。

3. Presto 引擎的 CSS3 径向渐变语法

Presto 引擎的 CSS3 径向渐变语法和 Webkit 引擎的新式语法类似,只是使用的前缀不同而已。

```
-o-radial-gradient([<position> || <angle>],)?[<shape> ||  
    <size>],]?<color-stop>,<color-stop>[,<color-stop>]*);
```

Presto 引擎中的 Opera 11.6 开始支持径向渐变。

4. Trident 引擎的 CSS3 径向渐变语法

Trident 引擎的浏览器从 IE 10 开始支持径向渐变语法，其语法格式与 Webkit 引擎的新式语法类似，不同的仅是其前缀不同，需要使用“-ms-”。

```
-ms-radial-gradient([<position> || <angle>],)?[<shape> ||  
    <size>],]?<color-stop>,<color-stop>[,<color-stop>]*);
```

5. W3C 标准径向渐变语法

W3C 组织从 2013 年 4 月开始给径向语法推出新的语法规则。

```
radial-gradient([[<shape> || <size>] [at <position>]?,  
    | at <position>],)?<color-stop>[,<color-stop>]+);
```

不过支持径向渐变语法的浏览器，到写这篇稿时，仅有 IE 10+、Firefox 16+ 浏览器支持。

10.3.2 CSS3 径向渐变的属性参数

CSS3 的径向渐变相对于线性渐变要复杂得多，属性参数也繁多复杂。CSS3 径向变中新旧语法中的属性参数定义如下。

1. <position>

主要用来定义径向渐变的圆心位置。此值类似于 CSS 中 background-position 属性，用于确定元素渐变的中心位置。如果这个参数省略了，其默认值为 center。其值主要有以下几种。

- ❑ <length>：用长度值指定径向渐变圆心的横坐标或纵坐标，可以为负值。
- ❑ <percentage>：用百分比指定径向渐变圆心的横坐标或纵坐标，可以为负值。
- ❑ left：设置左边为径向渐变圆心的横坐标值。
- ❑ center：设置中间为径向渐变圆心的横坐标值或纵坐标。
- ❑ right：设置右边为径向渐变圆心的横坐标值。
- ❑ top：设置顶部为径向渐变圆心的纵坐标值。
- ❑ bottom：设置底部为径向渐变圆心的纵坐标值。

2. <shape>

主要用来定义径向渐变的形状。其主要包括两个值 circle 和 ellipse：

- ❑ circle：如果 <size> 和 <length> 大小相等，径向渐变是一个圆形，也就是用来指定圆形的径向渐变。
- ❑ ellipse：如果 <size> 和 <length> 大小不相等，径向渐变是一个椭圆形，也就是用来指定椭圆形的径向渐变。

3. <size>

用来确定径向渐变的结束形状大小。如果省略，其默认值为 `farthest-corner`。可以显式地设置一些关键词。

- ❑ `closest-side`: 指定径向渐变的半径长度为从圆心到离圆心最近的边。
- ❑ `closest-corner`: 指定径向渐变的半径长度为从圆心到离圆心最近的角。
- ❑ `farthest-side`: 指定径向渐变的半径长度为从圆心到离圆心最远的边。
- ❑ `farthest-corner`: 指定径向渐变的半径长度为从圆心到离圆心最远的角。

如果 `<shape>` 设置为 `circle` 或者省略，`<size>` 可能显式设置为 `<length>`。表示用长度值指定径向渐变的横向或纵向直径长度，并根据横向和纵向的直径来确定径向渐变的形状是圆或者是椭圆，不能为负值。



注意 这里不能为百分比值，它们只用来指定椭圆径向渐变的大小，而不是圆形渐变大小。

如果 `<shape>` 设置 `ellipse` 或者省略，`<size>` 可能显式设置为 `[<length>|<percentage>]`。主要用来设置椭圆的大小。第一个值代表椭圆的水平半径，第二个值代表垂直半径。这两个值除了使用 `<length>` 定义大小之外还可以使用 `<percentage>` 来定义这两半径大小。使用 `<percentage>` 定义值是相对于径向渐变容器的尺寸，同样不能为负值。

4. <color-stop>

径向渐变线上的停止颜色，类似于线性渐变的 `<color-stop>`，渐变线从中心点向右扩散。其中 `0%` 表示渐变线的起始点，`100%` 表示渐变线的与渐变容器相交结束的位置，而且其颜色停止可以定义一个负值。

10.3.3 CSS3 径向渐变的基本用法

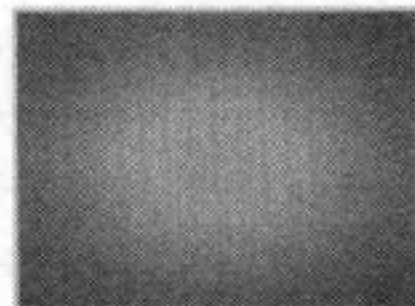
虽然径向渐变要比线性渐变更复杂，只要了解了其基本语法以及相关属性参数的作用。接下来，通过实例来加强径向渐变的使用。下面的所有例子都在一个宽度为 `400px`、高为 `300px` 的容器中实现。

1. 圆心相同

先看一个简单的径向渐变，圆心是容器正中间，从 `hsla(120,70%,60%,.9)` 向 `hsla(360,60%,60%,.9)` 颜色实现径向渐变。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title> 径向渐变 </title>
  <style type="text/css" media="screen">
    *{
      margin: 0;
```

```
padding: 0;
}
div {
width: 400px;
height: 300px;
margin: 50px auto;
border: 5px solid hsla(60,50%,80%,.8);
background-image: -webkit-radial-gradient(hsla(120,70%,60%,.9),
hsla(360,60%,60%,.9));
background-image: radial-gradient(hsla(120,70%,60%,.9),
hsla(360,60%,60%,.9));
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```



效果如图 10-21 所示。

☆图 10-21 制作同心圆径向渐变

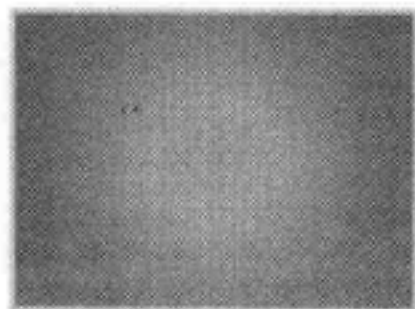
如果制作一个圆形渐变，只需要在前例的基础上添加一个关键词 `circle`。

```
div {
width: 400px;
height: 300px;
margin: 50px auto;
border: 5px solid hsla(60,50%,80%,.8);
background-image: -webkit-radial-gradient(circle,hsla(120,70%,60%,.9),
hsla(360,60%,60%,.9));
background-image: radial-gradient(circle,hsla(120,70%,60%,.9),
hsla(360,60%,60%,.9));
}
```

效果如图 10-22 所示。

圆形的渐变是一个特殊的椭圆渐变，水平半径和垂直半径具有相同的长度值，如图 10-23 所示。

圆形渐变是椭圆渐变的一种特殊情况，渐变主要半径（水平半径）和次要半径（垂直半径）不相同就是一个椭圆形渐变，如图 10-24 所示。



☆图 10-22 制作圆形渐变

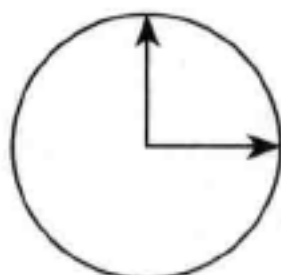


图 10-23 圆形渐变

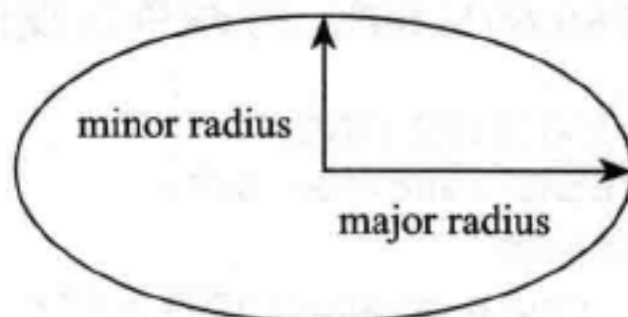


图 10-24 主要半径和次要半径不相等

在制作椭圆形渐变，可以使用关键词 `ellipse`。

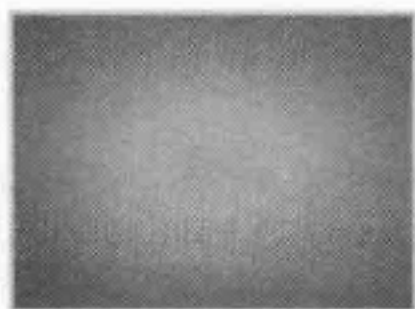
```
div {
    width: 400px;
    height: 300px;
    margin: 50px auto;
    border: 5px solid hsla(60,50%,80%,.8);
    background-image: -webkit-radial-gradient(ellipse,hsla(120,70%,60%,.9),
        hsla(360,60%,60%,.9));
    background-image: radial-gradient(ellipse,hsla(120,70%,60%,.9),
        hsla(360,60%,60%,.9));
}
```

效果如图 10-25 所示。

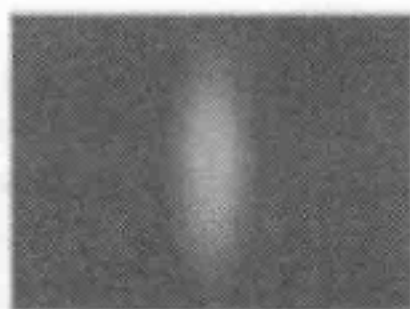
除了使用关键词制作不同的径向渐变，还可以用不同的渐变参数制作径向渐变效果，通过制作同心圆，主要半径和次要半径来决定径向渐变的形状。例如，圆心位置都在“200px,150px”处，半径为 50px 和 150px，从 hsla(120,70%,60%,.9) 到 hsla(360,60%,60%,.9) 径向渐变。

```
div {
    width: 400px;
    height: 300px;
    margin: 50px auto;
    border: 5px solid hsla(60,50%,80%,.8);
    background-image: -webkit-radial-gradient(50px 150px at 200px 150px,
        hsla(120,70%,60%,.9),hsla(360,60%,60%,.9));
    background-image: radial-gradient(50px 150px at 200px 150px,
        hsla(120,70%,60%,.9),hsla(360,60%,60%,.9));
}
```

效果如图 10-26 所示。实现的是内径小于外径制作的径向渐变效果。



☆图 10-25 通过 ellipse 制作椭圆形渐变



☆图 10-26 垂直椭圆径向渐变

接着来看圆心相同，内外半径大小相同实现的渐变效果。

```
div {
    width: 400px;
    height: 300px;
    margin: 50px auto;
    border: 5px solid hsla(60,50%,80%,.8);
    background-image: -webkit-radial-gradient(200px 200px at 200px 150px,
        hsla(120,70%,60%,.9),hsla(360,60%,60%,.9));
    background-image: radial-gradient(200px 200px at 200px 150px,
        hsla(120,70%,60%,.9),hsla(360,60%,60%,.9));
}
```


效果如图 10-27 所示。

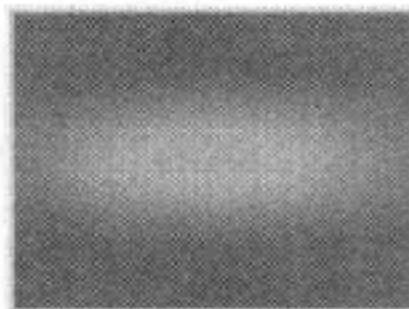
当内外圆的圆心相同，主要半径和次要半径相等时，等于制作一个圆形径向渐变效果。接下来再看一个实例，圆心相同，主要半径大于次要的半径制作的径向渐变。

```
div {
  width: 400px;
  height: 300px;
  margin: 50px auto;
  border: 5px solid hsla(60,50%,80%,.8);
  background-image: -webkit-radial-gradient(300px 100px at 200px 150px,
    hsla(120,70%,60%,.9),hsla(360,60%,60%,.9));
  background-image: radial-gradient(300px 100px at 200px 150px,
    hsla(120,70%,60%,.9),hsla(360,60%,60%,.9));
}
```

效果如图 10-28 所示。



☆图 10-27 圆心相同，内外半径相同



☆图 10-28 水平椭圆径向渐变

2. 椭圆形径向渐变

还可以使用 `<percentage>` 值，使用细节类似于 `<length>`。

```
div {
  width: 400px;
  height: 300px;
  margin: 50px auto;
  border: 5px solid hsla(60,50%,80%,.8);
  background-image: -webkit-radial-gradient(80% 20% at 30% 60%,
    hsla(120,70%,60%,.9),hsla(360,60%,60%,.9));
  background-image: radial-gradient(80% 20% at 30% 60%,
    hsla(120,70%,60%,.9),hsla(360,60%,60%,.9));
}
```

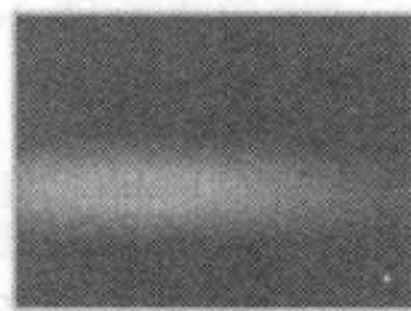
效果如图 10-29 所示。

3. 渐变形状配合圆心定位

除了上述方法能实现一些简的径向渐变效果之外，还可以使用渐变形状配合圆心定位。主要使用 `at` 加上关键词来定义径向渐变中心位置。径向渐变中心位置类似于 `background-position` 属性，可以使用一些关键词来定义。

(1) center

设置径向渐变中心位置在容器的中心点，相当于“`.at 50% 50%`”。类似于 `background-`



☆图 10-29 百分比制作椭圆形径向渐变

position 的 center。

```
/* at center*/
.center .circle {
    background-image: -webkit-radial-gradient(circle at center,
        rgb(220, 75, 200),rgb(0, 0, 75));
    background-image: radial-gradient(circle at center,
        rgb(220, 75, 200),rgb(0, 0, 75));
}
.center .ellipse {
    background-image: -webkit-radial-gradient(ellipse at center,
        rgb(220, 75, 200),rgb(0, 0, 75));
    background-image: radial-gradient(ellipse at center,
        rgb(220, 75, 200),rgb(0, 0, 75));
}
```

效果如图 10-30 所示。

(2) top

设置径向圆心在容器的顶边中心点处，与“at 50% 0%”等效。类似于 background-position 的 center top。

```
/* at top*/
.top .circle {
    background-image: -webkit-radial-gradient(circle at top,
        rgb(220, 75, 200),rgb(0, 0, 75));
    background-image: radial-gradient(circle at top,
        rgb(220, 75, 200),rgb(0, 0, 75));
}
.top .ellipse {
    background-image: -webkit-radial-gradient(ellipse at top,
        rgb(220, 75, 200),rgb(0, 0, 75));
    background-image: radial-gradient(ellipse at top,
        rgb(220, 75, 200),rgb(0, 0, 75));
}
```

效果如图 10-31 所示。



☆图 10-30 径向渐变圆心在元素容器正中间



☆图 10-31 径向渐变圆心在容器顶边中心点

(3) right

设置径向渐变圆心在容器右边中心点处，与“at 100% 50%”等效。类似于 background-position 的 right center。

```

/* at right */
.right .circle {
    background-image: -webkit-radial-gradient(circle at right,
        rgb(220, 75, 200),rgb(0, 0, 75));
    background-image: radial-gradient(circle at right,
        rgb(220, 75, 200),rgb(0, 0, 75));
}
.right .ellipse {
    background-image: -webkit-radial-gradient(ellipse at right,
        rgb(220, 75, 200),rgb(0, 0, 75));
    background-image: radial-gradient(ellipse at right,
        rgb(220, 75, 200),rgb(0, 0, 75));
}

```

效果如图 10-32 所示。

(4) bottom

设置径向渐变圆心在容器底边中心点处,刚好与 top 关键词位置相反,与“at 50% 100%”等效。类似于 background-position 中的 center bottom。

```

.bottom .circle {
    background-image: -webkit-radial-gradient(circle at bottom,
        rgb(220, 75, 200),rgb(0, 0, 75));
    background-image: radial-gradient(circle at bottom, rgb(220, 75, 200),rgb(0, 0, 75));
}
.bottom .ellipse {
    background-image: -webkit-radial-gradient(ellipse at bottom,
        rgb(220, 75, 200),rgb(0, 0, 75));
    background-image: radial-gradient(ellipse at bottom,
        rgb(220, 75, 200),rgb(0, 0, 75));
}

```

效果如图 10-33 所示。



☆图 10-32 径向渐变圆心在容器右边中心点



☆图 10-33 径向渐变圆心在容器底边中心点

(5) left

设置径向渐变圆心点在容器左边中心点处,刚好与 right 关键词位置相反,与“at 0% 50%”等效。类似于 background-position 的 center left。

```

/* at left */
.left .circle {
    background-image: -webkit-radial-gradient(circle at left,
        rgb(220, 75, 200),rgb(0, 0, 75));

```



```

background-image: radial-gradient(circle at left,
                                rgb(220, 75, 200),rgb(0, 0, 75));
}
.left .ellipse {
background-image: -webkit-radial-gradient(ellipse at left,
                                rgb(220, 75, 200),rgb(0, 0, 75));
background-image: radial-gradient(ellipse at left,
                                rgb(220, 75, 200),rgb(0, 0, 75));
}

```

效果如图 10-34 所示。

(6) top left

设置径向渐变圆心点在容器的左角顶点处,与关键词“left top”和“at 0 0”等效。类似于 background-position 的 left top。



☆图 10-34 径向渐变圆心在容器左边中心点

```

/* at top left*/
.top-left .circle {
background-image: -webkit-radial-gradient(circle at top left,
                                rgb(220, 75, 200),rgb(0, 0, 75));
background-image: radial-gradient(circle at top left,
                                rgb(220, 75, 200),rgb(0, 0, 75));
}
.top-left .ellipse {
background-image: -webkit-radial-gradient(ellipse at top left,
                                rgb(220, 75, 200),rgb(0, 0, 75));
background-image: radial-gradient(ellipse at top left,
                                rgb(220, 75, 200),rgb(0, 0, 75));
}
/* at left top*/
.left-top .circle {
background-image: -webkit-radial-gradient(circle at left top,
                                rgb(220, 75, 200),rgb(0, 0, 75));
background-image: radial-gradient(circle at left top,
                                rgb(220, 75, 200),rgb(0, 0, 75));
}
.left-top .ellipse {
background-image: -webkit-radial-gradient(ellipse at left top,
                                rgb(220, 75, 200),rgb(0, 0, 75));
background-image: radial-gradient(ellipse at left top,
                                rgb(220, 75, 200),rgb(0, 0, 75));
}

```

效果如图 10-35 所示。

(7) top right

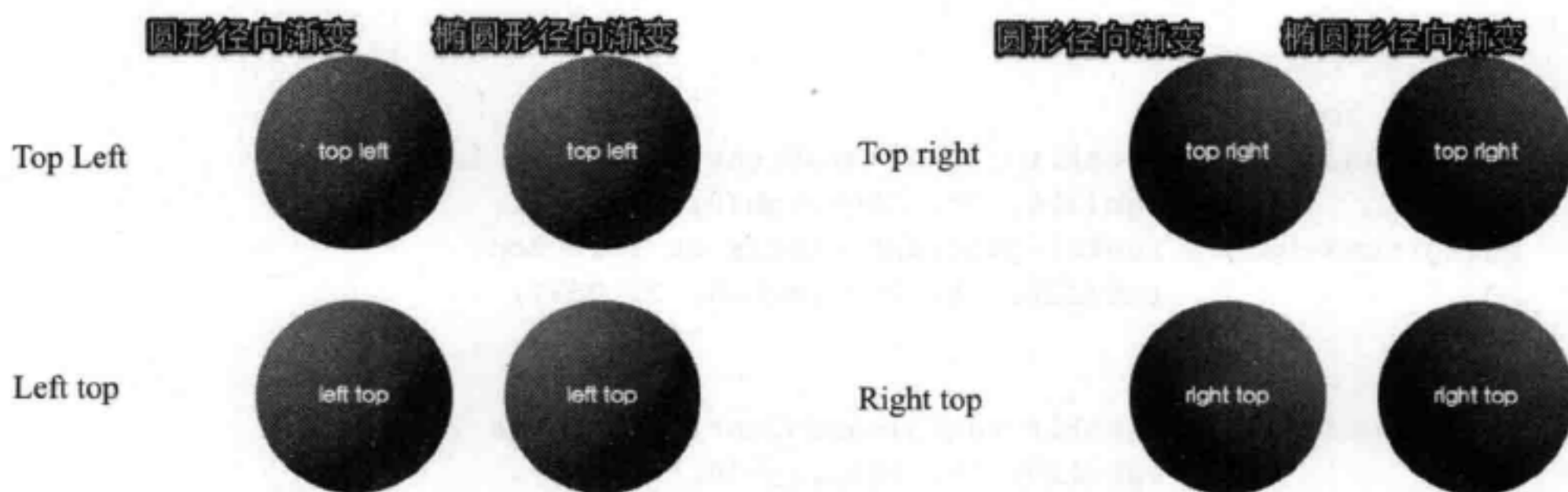
设置径向渐变圆心点在容器右角顶点处,与“right top”关键词和“at 100% 0%”等效。类似于 background-position 的 top right。

```

/*at top right*/
.top-right .circle {
    background-image: -webkit-radial-gradient(circle at top right,
        rgb(220, 75, 200),rgb(0, 0, 75));
    background-image: radial-gradient(circle at top right,
        rgb(220, 75, 200),rgb(0, 0, 75));
}
.top-right .ellipse {
    background-image: -webkit-radial-gradient(ellipse at top right,
        rgb(220, 75, 200),rgb(0, 0, 75));
    background-image: radial-gradient(ellipse at top right,
        rgb(220, 75, 200),rgb(0, 0, 75));
}
/* at right top*/
.right-top .circle {
    background-image: -webkit-radial-gradient(circle at right top,
        rgb(220, 75, 200),rgb(0, 0, 75));
    background-image: radial-gradient(circle at right top,
        rgb(220, 75, 200),rgb(0, 0, 75));
}
.right-top .ellipse {
    background-image: -webkit-radial-gradient(ellipse at right top,
        rgb(220, 75, 200),rgb(0, 0, 75));
    background-image: radial-gradient(ellipse at right top,
        rgb(220, 75, 200),rgb(0, 0, 75));
}

```

效果如图 10-36 所示。



☆图 10-35 径向渐变圆心在容器左上角顶点

☆图 10-36 径向渐变圆心点在容器右角顶点

(8) bottom right

设置径向渐变的圆心点在容器右下角顶点处，与关键词“right bottom”和“at 100% 100%”等效。类似于 background-position 的 bottom right。

```

/* at bottom right*/
.bottom-right .circle {

```

```

background-image: -webkit-radial-gradient(circle at bottom right,
    rgb(220, 75, 200),rgb(0, 0, 75));
background-image: radial-gradient(circle at bottom right,
    rgb(220, 75, 200),rgb(0, 0, 75));
}
.bottom-right .ellipse {
    background-image: -webkit-radial-gradient(ellipse at bottom right,
        rgb(220, 75, 200),rgb(0, 0, 75));
    background-image: radial-gradient(ellipse at bottom right,
        rgb(220, 75, 200),rgb(0, 0, 75));
}
/* at right bottom*/
.right-bottom .circle {
    background-image: -webkit-radial-gradient(circle at right bottom,
        rgb(220, 75, 200),rgb(0, 0, 75));
    background-image: radial-gradient(circle at right bottom,
        rgb(220, 75, 200),rgb(0, 0, 75));
}
.right-bottom .ellipse {
    background-image: -webkit-radial-gradient(ellipse at right bottom,
        rgb(220, 75, 200),rgb(0, 0, 75));
    background-image: radial-gradient(ellipse at right bottom,
        rgb(220, 75, 200),rgb(0, 0, 75));
}

```

效果如图 10-37 所示。

(9) bottom left

设置径向渐变圆心在容器左下角顶点处，与关键词“left bottom”和“at 0% 100%”等效。类似于 background-position 的 bottom left。

```

/* at bottom left*/
.bottom-left .circle {
    background-image: -webkit-radial-gradient(circle at bottom left,
        rgb(220, 75, 200),rgb(0, 0, 75));
    background-image: radial-gradient(circle at bottom left,
        rgb(220, 75, 200),rgb(0, 0, 75));
}
.bottom-left .ellipse {
    background-image: -webkit-radial-gradient(ellipse at bottom left,
        rgb(220, 75, 200),rgb(0, 0, 75));
    background-image: radial-gradient(ellipse at bottom left,
        rgb(220, 75, 200),rgb(0, 0, 75));
}
/* at left bottom*/
.left-bottom .circle {
    background-image: -webkit-radial-gradient(circle at left bottom,
        rgb(220, 75, 200),rgb(0, 0, 75));
    background-image: radial-gradient(circle at left bottom,
        rgb(220, 75, 200),rgb(0, 0, 75));
}

```

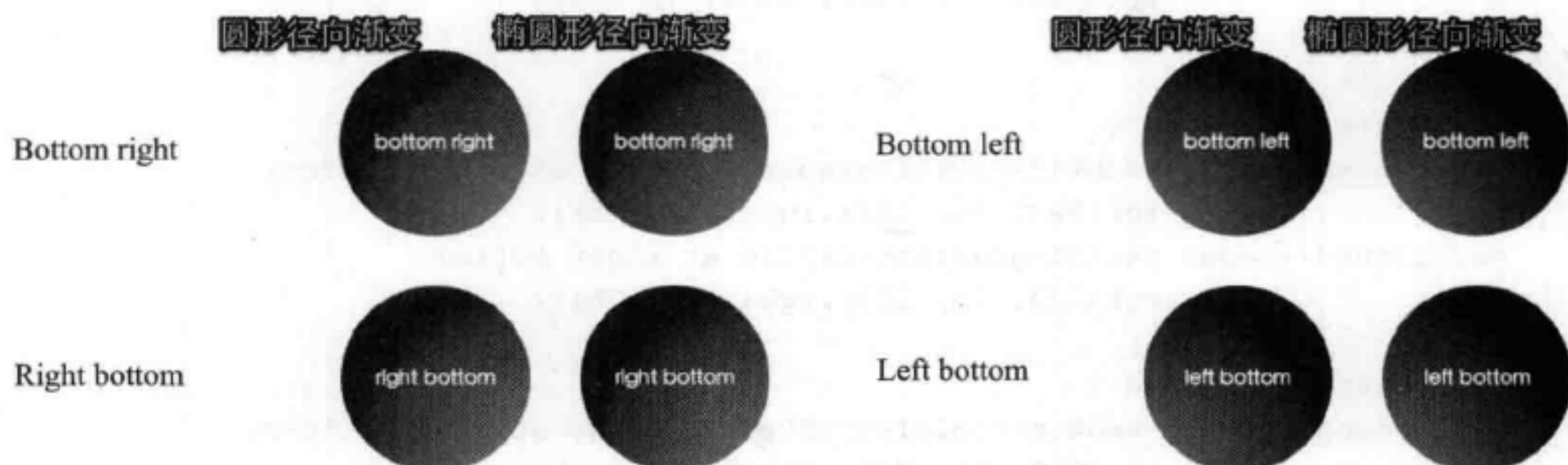


```

}
.left-bottom .ellipse {
    background-image: -webkit-radial-gradient(ellipse at left bottom,
        rgb(220, 75, 200),rgb(0, 0, 75));
    background-image: radial-gradient(ellipse at left bottom,
        rgb(220, 75, 200),rgb(0, 0, 75));
}

```

效果如图 10-38 所示。



☆图 10-37 径向渐变圆心在容器右下角顶点

☆图 10-38 径向渐变圆心点在容器左下角顶点

综合上面的实例以及对应的效果，大家在理解径向渐变使用关键词设置径向渐变圆心位置，可以把其当作元素背景中的 background-position 属性来理解。用来设置元素径向渐变圆心的所有可用关键词对照的位置关系，可以浏览图 10-39。

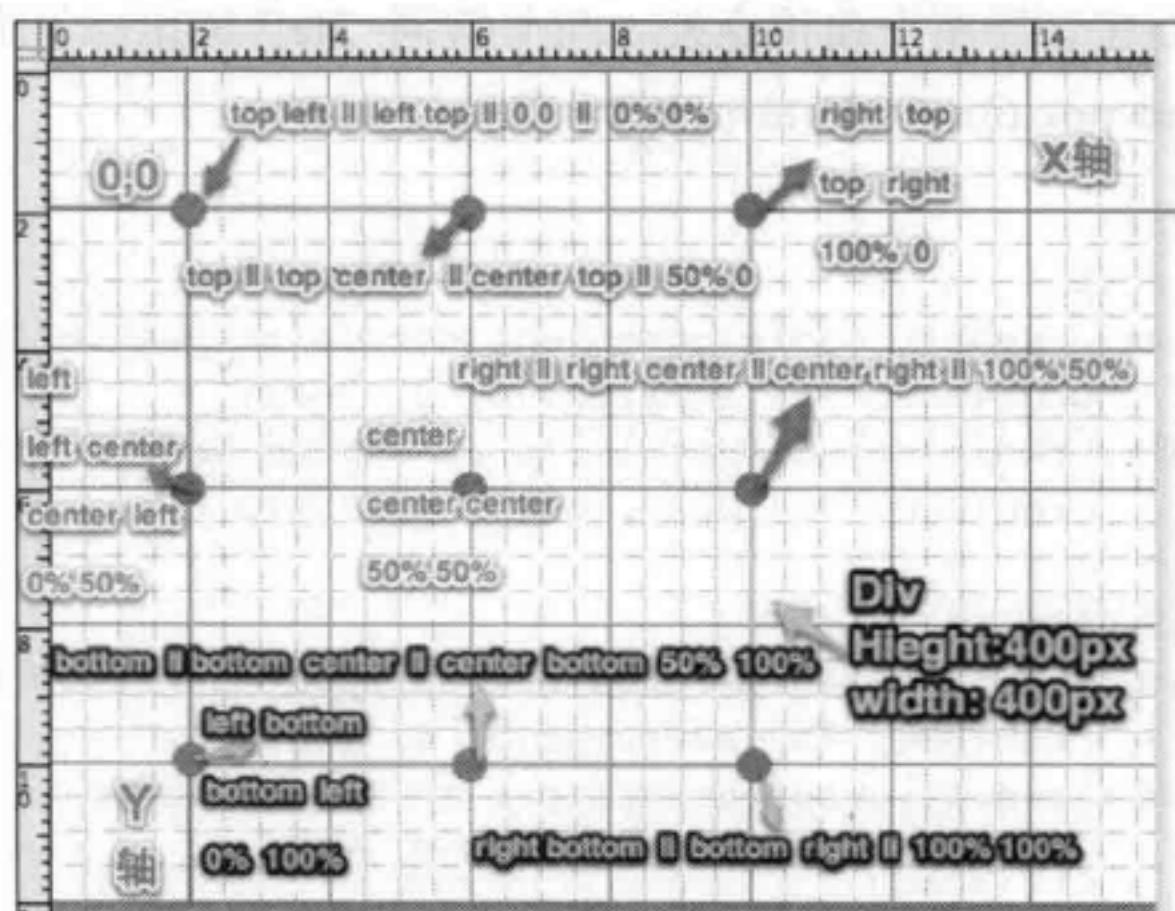


图 10-39 设置径向渐变圆心位置关键词对照

前面分别介绍了使用 <size>、渐变类型关键词以及渐变类型配合关键词制作的径向渐变效果。除了上述方法之外，还可以使用 <size> 和渐变类型以及圆心关键词结合制作一些径向渐变效果。

当给元素显式设置了径向渐变的大小,这两个值设置了径向渐变的水平和垂直半径。如果一个径向渐变包含径向渐变类型 `circle` 和单一的大小值,就可以实现一个圆形的径向渐变。如果一个径向渐变包含了径向渐变类型 `ellipse` 和两个值,或者只是两个值,就可以实现一个椭圆形的径向渐变。在设置径向半径大小值时,可以使任何 CSS 的单位长度值都有效。

1) 制作一个径向半径为 20px 的圆形径向渐变。

```
div {
    width: 300px;
    height: 300px;
    border-radius: 100%;
    margin: 50px auto;
    border: 5px solid hsla(60,50%,80%,.8);
    background-image: -webkit-radial-gradient(20px circle,
        hsla(220,89%,100%,1),hsla(30,60%,60%,.9));
    background-image: radial-gradient(20px circle,
        hsla(220,89%,100%,1),hsla(30,60%,60%,.9));
}
```

效果如图 10-40 所示。

2) 制作椭圆形径向渐变,其中主要半径为 2em,次要半径为 4em。

```
div {
    width: 300px;
    height: 300px;
    border-radius: 100%;
    margin: 50px auto;
    border: 5px solid hsla(60,50%,80%,.8);
    background-image: -webkit-radial-gradient(2em 4em ellipse,
        hsla(220,89%,100%,1),hsla(30,60%,60%,.9));
    background-image: radial-gradient(2em 4em ellipse,
        hsla(220,89%,100%,1),hsla(30,60%,60%,.9));
}
```

效果如图 10-41 所示。

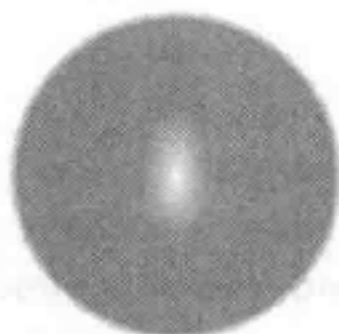
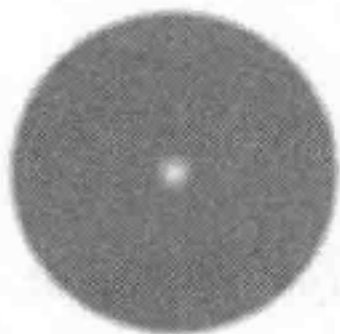


图 10-40 一个长度值和 `circle` 制作圆形径向渐变 图 10-41 `size` 值配合 `ellipse` 制作椭圆形径向渐变

3) 制作一个圆形径向渐变,其半径为 5em,圆心在 top。

```
div {
    width: 300px;
    height: 300px;
```

```
border-radius: 100%;
margin: 50px auto;
background-image: -webkit-radial-gradient(8em circle at top,
hsla(220,89%,100%,1),hsla(30,60%,60%,.9));
background-image: radial-gradient(8em circle at top,
hsla(220,89%,100%,1),hsla(30,60%,60%,.9));
}
```



效果如图 10-42 所示。

图 10-42 8em circle at top 制作圆形径向渐变

4. 关键词设置

还可以通过关键词隐式为径向渐变设置大小，其中的每个关键词指定径向渐变大小的算法。也就是通过圆心指向径向渐变的边或者角来确定径向渐变的大小。不过在圆形和椭圆形的径向渐变之中，算出来的大小略有不同。虽然最初看上去似乎有些复杂，一旦理解了这些关键词的意义，一切就变得简单，如表 10-3 所示。

表 10-3 关键词设置径向渐变

关键词	圆形径向渐变		椭圆形径向渐变	
closest-side	径向边缘与离元素最近边缘的切点到圆心之间的直线距离大决定		径向渐变水平与垂直边缘与离最近边缘切点到圆心的直线距离大小决定	
closest-corner	渐变边缘与到元素最近顶角交接点到圆心直线距离决定		径向渐变边缘与到元素最近顶角交接点到圆心的直线距离大小决定	
farthest-side	径向渐变边缘与元素最远边的切点到圆心的直线距离大小决定		径向渐变与元素最远边切点到圆心的直线距离大小决定	
farthest-corner	径向渐变边缘与元素最远顶角交点到圆心的直线距离决定		渐变边缘与元素最远顶点的交接点到圆心的直线距离决定	

为了更好的说明这些关键词的使用，下面的实例中，将圆心定义在“at 50% 75%”位置，分别演示 closest-side、closest-corner、farthest-side 和 farthest-corner 在圆形与椭圆形径向渐变的效果。

```
/* closest-side*/
.closest-side .circle {
background-image: -webkit-radial-gradient(closest-side circle at 50% 75%,
rgb(220, 75, 200),rgb(0, 0, 75));
background-image: radial-gradient(closest-side circle at 50% 75%,
rgb(220, 75, 200),rgb(0, 0, 75));
}
.closest-side .ellipse {
background-image: -webkit-radial-gradient(closest-side ellipse at 50% 75%,
rgb(220, 75, 200),rgb(0, 0, 75));
background-image: radial-gradient(closest-side ellipse at 50% 75%,
```



```

        rgb(220, 75, 200),rgb(0, 0, 75));
    }
    /* closest-corner*/
    .closest-corner .circle {
        background-image: -webkit-radial-gradient(closest-corner circle at 50% 75%,
            rgb(220, 75, 200),rgb(0, 0, 75));
        background-image: radial-gradient(closest-corner circle at 50% 75%,
            rgb(220, 75, 200),rgb(0, 0, 75));
    }
    .closest-corner .ellipse {
        background-image: -webkit-radial-gradient(closest-corner at 50% 75%,
            rgb(220, 75, 200),rgb(0, 0, 75));
        background-image: radial-gradient(closest-corner ellipse at 50% 75%,
            rgb(220, 75, 200),rgb(0, 0, 75));
    }
    /* farthest-side */
    .farthest-side .circle {
        background-image: -webkit-radial-gradient(farthest-side circle at 50% 75%,
            rgb(220, 75, 200),rgb(0, 0, 75));
        background-image: radial-gradient(farthest-side circle at 50% 75%,
            rgb(220, 75, 200),rgb(0, 0, 75));
    }
    .farthest-side .ellipse {
        background-image: -webkit-radial-gradient(farthest-side ellipse at 50% 75%,
            rgb(220, 75, 200),rgb(0, 0, 75));
        background-image: radial-gradient(farthest-side ellipse at 50% 75%,
            rgb(220, 75, 200),rgb(0, 0, 75));
    }
    /* farthest-corner */
    .farthest-corner .circle {
        background-image: -webkit-radial-gradient(farthest-corner circle at 50% 75%,
            rgb(220, 75, 200),rgb(0, 0, 75));
        background-image: radial-gradient(farthest-corner circle at 50% 75%,
            rgb(220, 75, 200),rgb(0, 0, 75));
    }
    .farthest-corner .ellipse {
        background-image: -webkit-radial-gradient(farthest-corner ellipse at 50% 75%,
            rgb(220, 75, 200),rgb(0, 0, 75));
        background-image: radial-gradient(farthest-corner ellipse at 50% 75%,
            rgb(220, 75, 200),rgb(0, 0, 75));
    }
}

```

效果如图 10-43 所示。

在径向渐变中，除了能设置径向渐变的圆心位置、半径大小之外，还可以设置径向渐变的颜色，前面演示的都是简单的两个颜色制作径向渐变，接下来通过 <color-stop> 属性参数来设置径向渐变的颜色。

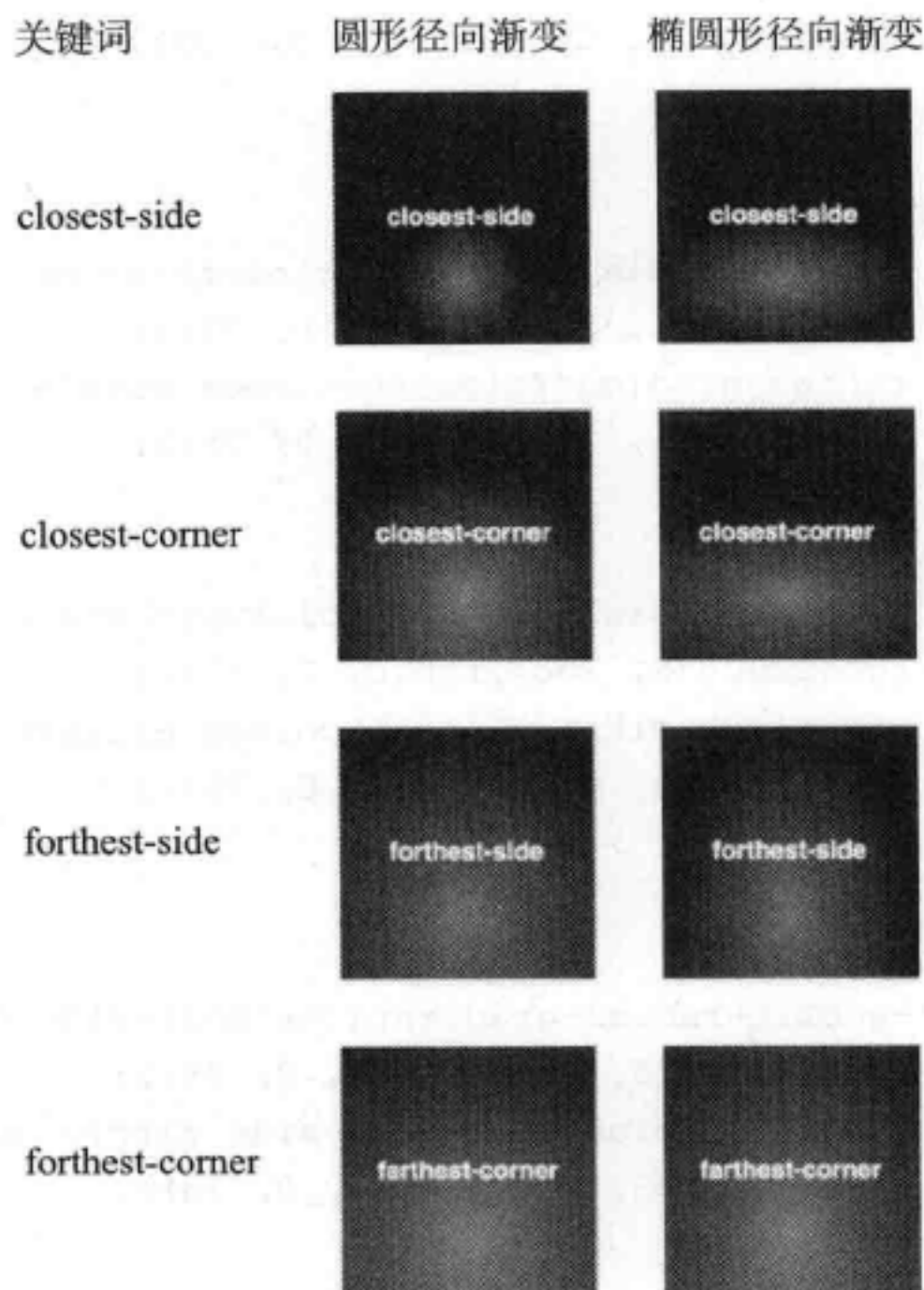


图 10-43 关键词制作径向渐变效果

5. 三色渐变

在前面的基础上增加一个颜色，实现三色渐变效果。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title> 径向渐变——三色径向渐变 </title>
  <style type="text/css">
    *{
      margin: 0;
      padding: 0;
    }
    div {
      margin: 20px auto;
      width: 200px;
      height: 200px;
      border-radius: 100%;
      background-image: -webkit-radial-gradient (red, green, blue);
      background-image: radial-gradient (red, green, blue);
    }
  </style>
</head>
```

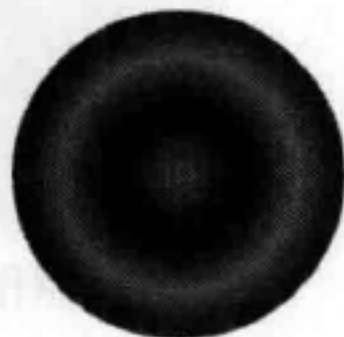
```
<body>
  <div></div>
</body>
</html>
```

效果如图 10-44 所示。

上面的示例是一个简单的三色径向渐变，只是通过设置三个颜色，从容器的中心向外由红色（red）、绿色（green）到蓝色（blue）。可以说这是最简单的多色径向渐变了，此外，还可以给每个颜色设置具体的显示位置。

```
div {
  margin: 20px auto;
  width: 200px;
  height: 200px;
  border-radius: 100%;
  background-image: -webkit-radial-gradient(red 20%,green 50%,blue 80%);
  background-image: radial-gradient(red 20%,green 50%,blue 80%);
}
```

效果如图 10-45 所示。



☆图 10-44 三色径向渐变



☆图 10-45 配有具体色标位置的三色径向渐变

在径向渐变中的渐变颜色，可以使用任何表示颜色的格式，确定渐变颜色的位置使用任何表示长度的单位值，同时颜色数量不会做任何限制，而且前面介绍有关于径向渐变的属性参数都可以配合多种颜色，制作出一些更特殊的径向渐变效果。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title> 径向渐变——制作多色径向渐变 </title>
  <style type="text/css">
    *{
      margin: 0;
      padding: 0;
    }
    div {
      margin: 20px auto;
      width: 300px;
      height: 200px;
    }
  </style>
</head>
<body>
```



```

.circle {
background-image: -webkit-radial-gradient(90px circle at top,
rgb(0,0,0) 30%,rgba(255,190,90,.9) 200px,hsl(123,58%,90%) 8em,
hsla(230,40%,90%,.8) 80%,blue);
background-image: radial-gradient(90px circle at top,rgb(0,0,0) 30%,
rgba(255,190,90,.9) 200px,hsl(123,58%,90%) 8em,
hsla(230,40%,90%,.8) 80%,blue);
}
.ellipse {
background-image: -webkit-radial-gradient(
farthest-side ellipse at top,
rgb(0,0,0) 30%,rgba(255,190,90,.9) 200px,
hsl(123,58%,90%) 8em,hsla(230,40%,90%,.8) 80%,blue);
background-image: radial-gradient(
farthest-side ellipse at top,
rgb(0,0,0) 30%,rgba(255,190,90,.9) 200px,
hsl(123,58%,90%) 8em,
hsla(230,40%,90%,.8) 80%,blue);
}
</style>
</head>
<body>
<div class="circle"></div>
<div class="ellipse"></div>
</body>
</html>

```

效果如图 10-46 所示。

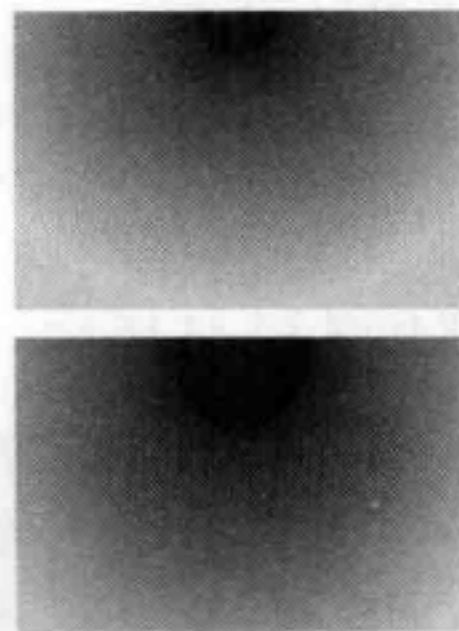


图 10-46 制作多色不同位置
径向渐变效果

10.3.4 实战体验：CSS3 径向渐变制作圆形图标按钮

在线性渐变一节中，利用所学的知识制作了一套带有 icon 的按钮，接下来结合前面的 text-shadow、box-shadow、多背景等有关属性以及后面将要介绍的 @font-face 属性制作圆形图标按钮，如图 10-47 所示。



☆图 10-47 CSS3 径向渐变制作圆形
图标按钮

在图 10-47 的效果中，主要用了三个 button 标签。

```

<button type="button" class="button">Chrome</button>
<button type="button" class="button">Firefox</button>
<button type="button" class="button">IE</button>

```

从效果中可以分析得出，整个背景使用了径向渐变、多背景等属性来实现斑点纹理背景，其次使用径向渐变配合圆角属性 border-radius 实现按钮背景，最后使用 @font-face 实现 icon 效果。先来看斑点纹理背景实现的代码。

```

body{
background-color: #282828;

```

```
background-image:
  -webkit-radial-gradient(black 15%, transparent 16%),
  -webkit-radial-gradient(black 15%, transparent 16%),
  -webkit-radial-gradient(rgba(255, 255, 255, 0.1) 15%, transparent 20%),
  -webkit-radial-gradient(rgba(255, 255, 255, 0.1) 15%, transparent 20%);
background-image:
  radial-gradient(black 15%, transparent 16%),
  radial-gradient(black 15%, transparent 16%),
  radial-gradient(rgba(255, 255, 255, 0.1) 15%, transparent 20%),
  radial-gradient(rgba(255, 255, 255, 0.1) 15%, transparent 20%);
background-position: 0 0px, 8px 8px, 0 1px, 8px 9px;
background-size: 16px 16px;
}
```

在整个斑点纹理背景制作中,使用了四个径向圆形渐变制作不同的四张背景图片,并定位在不同的位置,并且改变每张背景图片尺寸平铺在整个 body 容器之中、最终效果如图 10-48 所示。

完成背景之后,使用径向渐变、圆角属性和盒子阴影实现按钮背景。

```
.button{
  width: 70px;
  height: 70px;
  margin-right: 90px;
  font-size: 0;
  border-radius: 50%;
  border: none;
  box-shadow:
    0 1px 5px rgba(255,255,255,.5) inset,
    0 -2px 5px rgba(0,0,0,.3) inset,
    0 3px 8px rgba(0,0,0,.8);
  background: -webkit-radial-gradient( circle at top center, #f28fb8, #e982ad, #ec568c);
  background: radial-gradient(circle at top center, #f28fb8, #e982ad, #ec568c);
}
```

效果如图 10-49 所示。



☆图 10-48 斑点纹理背景



☆图 10-49 按钮背景

现在离整个效果只差一步,就是实现按钮的图标效果。在这个示例中,采用 @font-face 属性配合伪类 “:after” 制作图标。为了让整个图标的视觉效果更完美一些,采用 text-shadow 属性为图标制作发光效果。

```

/* 调用本地字体 */
@font-face {
  font-family: 'icomoon';
  src:url('font/icomoon.eot');
  src:url('font/icomoon.eot?#iefix') format('embedded-opentype'),
  url('font/icomoon.svg#icomoon') format('svg'),
  url('font/icomoon.woff') format('woff'),
  url('font/icomoon.ttf') format('truetype');
  font-weight: normal;
  font-style: normal;
}
.button:nth-child(3){
  margin-right: 0;
}
.button:after{
  font-family: 'icomoon';
  speak: none;
  font-weight: normal;
  -webkit-font-smoothing: antialiased;
  font-size: 36px;
  /* 制作 chrome 图标 */
  content: "\21";
  color: #dd5183;
  /* 制作图标发光效果 */
  text-shadow:0 3px 10px #f1a2c1,0 -3px 10px #f1a2c1;
}
/* 制作 firefox 图标 */
.button:nth-child(2):after{
  content: "\22";
}
/* 制作 ie 图标 */
.button:nth-child(3):after{
  content: "\23";
}

```

这个时候整个按钮效果如图 10-47 所示。为了让用户体验更完美些，可以给按钮添加悬停状态和点击状态下效果。

```

.button:hover:after{
  color: #fff;
  text-shadow:0 1px 20px #fccdda, 1px 0 14px #fccdda;
}
.button:active{
  box-shadow:
    0 2px 7px rgba(0,0,0,.5) inset,
    0 -3px 10px rgba(0,0,0,.1) inset,
    0 1px 3px rgba(255,255,255,.5);
  background: -webkit-radial-gradient(circle at top center,
    #f28fb8, #e982ad, #ec568c);
  background: radial-gradient(circle at top center, #f28fb8, #e982ad, #ec568c);
}

```


效果如图 10-50 所示。

到这，CSS3 径向渐变配合其他属性制作的圆形按钮就完成了，是整个实例的代码可从网址下载。



图 10-50 按钮悬浮状态

10.4 CSS3 重复渐变

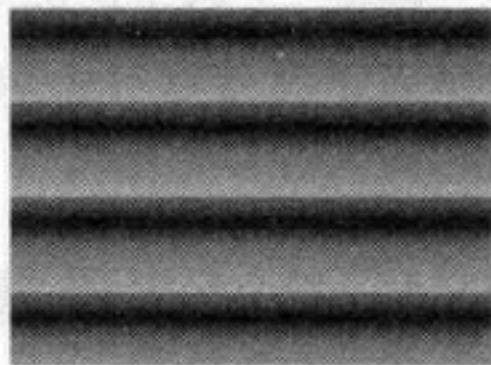
线性渐变和径向渐变都属于 CSS 背景属性中的背景图片（background-image）属性。有时候希望创建一个元素的背景上重复的渐变“模式”。在没有重复渐变的属性之前，主要通过重复背景图像（使用 background-repeat）创建线性重复渐变，但是没有创建重复的径向渐变的类似方式。幸运的是，CSS3 通过 repeating-linear-gradient 和 repeating-radial-gradient 语法提供了补救方法，可以直接实现重复的渐变效果。

10.4.1 CSS3 重复线性渐变

可以使用重复线性渐变 repeating-linear-gradient 属性代替线性渐变 linear-gradient。它们采取相同的值，但色标在两个方向上都无限重复。不过使用百分比设置色标的位置没有多大的意义，但使用像素和其他的单位重复线性渐变可以创建一些很酷的效果，例如下面的实例。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title> 重复线性渐变 </title>
  <style type="text/css">
    *{
      margin: 0;
      padding: 0;
    }
    div {
      width: 400px;
      height: 300px;
      margin: 20px auto;
      background-image: -webkit-repeating-linear-gradient(
        red,green 40px, orange 80px);
      background-image: repeating-linear-gradient(
        red,green 40px, orange 80px);
    }
  </style>
</head>
<body>
  <div></div>
</body>
</html>
```

在这个例子中从开始的红色 (red) 向 40px 处绿色 (green) 渐变, 然后向 80px 处橙色 (orange) 渐变。因为是一个重复的线性渐变, 不断以这个模式从上向下重复平铺, 如图 10-51 所示。

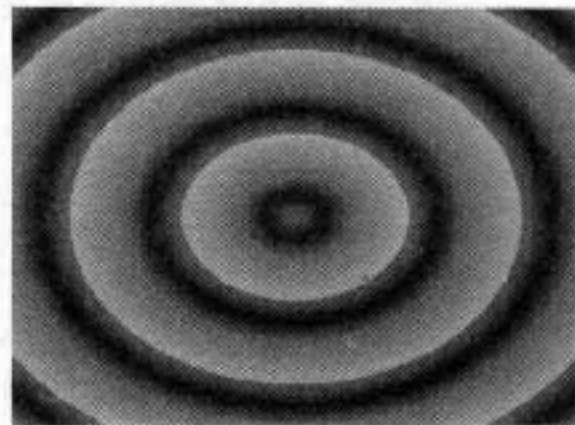


☆图 10-51 重复线性渐变

10.4.2 CSS3 重复径向渐变

以同样的方式, 使用相关的属性创建重复的径向渐变, 其语法和 radial-gradient 类似, 只是以一个径向渐变为基础进行重复渐变, 如下所示。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title> 重复径向渐变 </title>
  <style type="text/css">
    *{
      margin: 0;
      padding: 0;
    }
    div {
      width: 400px;
      height: 300px;
      margin: 20px auto;
      background-image: -webkit-radial-linear-gradient(
        red,green 40px, orange 80px);
      background-image: repeating-radial-gradient(
        red,green 40px, orange 80px);
    }
  </style>
</head>
<body>
  <div></div>
</body>
</html>
```



☆图 10-52 重复径向渐变

效果如图 10-52 所示。

10.4.3 实战体验: 制作记事本纸张效果

记事本大家都清楚, 每张纸都有横线条, 左边有两条竖线从顶部延伸到底部。今天使用重复渐变制作这样的纸张背景效果。不使用任何图片, 只使用 CSS3 的重复渐变在 body 上实现效果。

```
html,
body {
  margin: 0;
```

```
padding: 0;
height: 100%;
}
body {
background: -webkit-repeating-linear-gradient(to top,
      #f9f9f9, #f9f9f9 29px, #ccc 30px);
background: repeating-linear-gradient( to top,
      #f9f9f9, #f9f9f9 29px, #ccc 30px );
background-size: 100% 30px;
position: relative;
}
```

在这个效果中,仅重复渐变属性是无法完成的,还需要 CSS3 的另一个属性 background-size,用来指定背景图像的大小为 100% 30px。尽管 CSS3 渐变显示的是颜色,实际上它是一张图像而不是颜色,如图 10-53 所示。

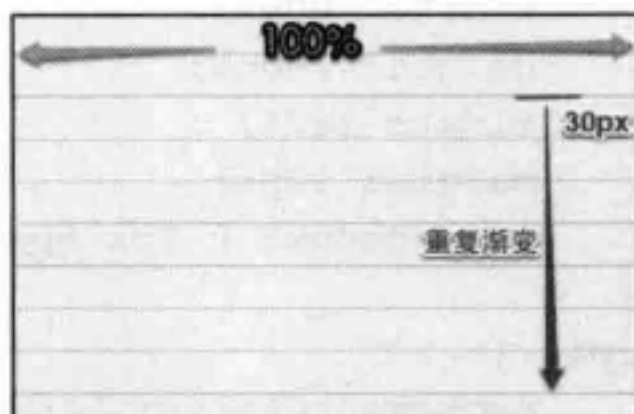


图 10-53 重复渐变制作线条平铺效果

接下来使用伪类“:before”在纸张边添加两条竖线,如图 10-54 所示。

```
body:before {
content: "";
display: inline-block;
height: 100%;
width: 4px;
border-left: 4px double #FCA1A1;
position: absolute;
left: 30px;
}
```

最终看到的效果如图 10-55 所示。

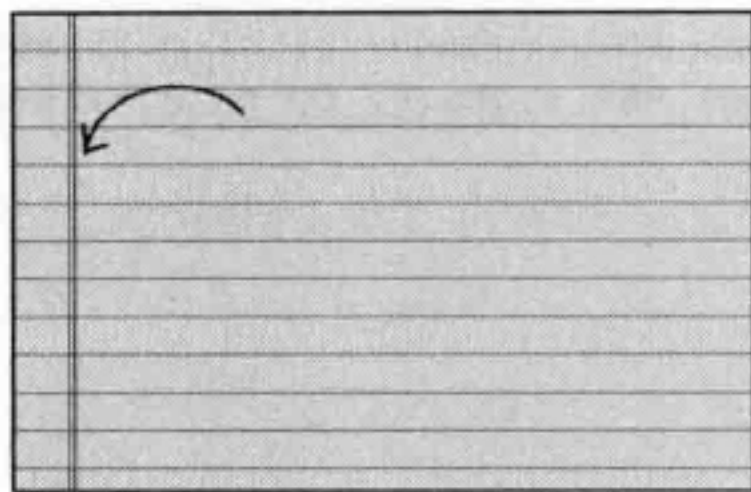
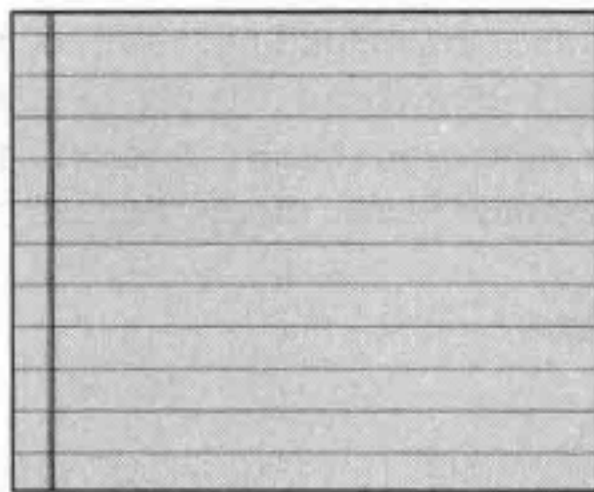


图 10-54 伪类制作竖线图



10-55 重复渐变制作记事本纸张效果

10.5 综合案例: CSS3 渐变制作纹理背景

Web 页面中常用纹理图片制作背景,这通过制图软件很快就能实现,但对于不懂设计的人来说并不是一件易事。CSS3 的渐变特性的出现,可以直接使用代码实现一些纹理背景效果。

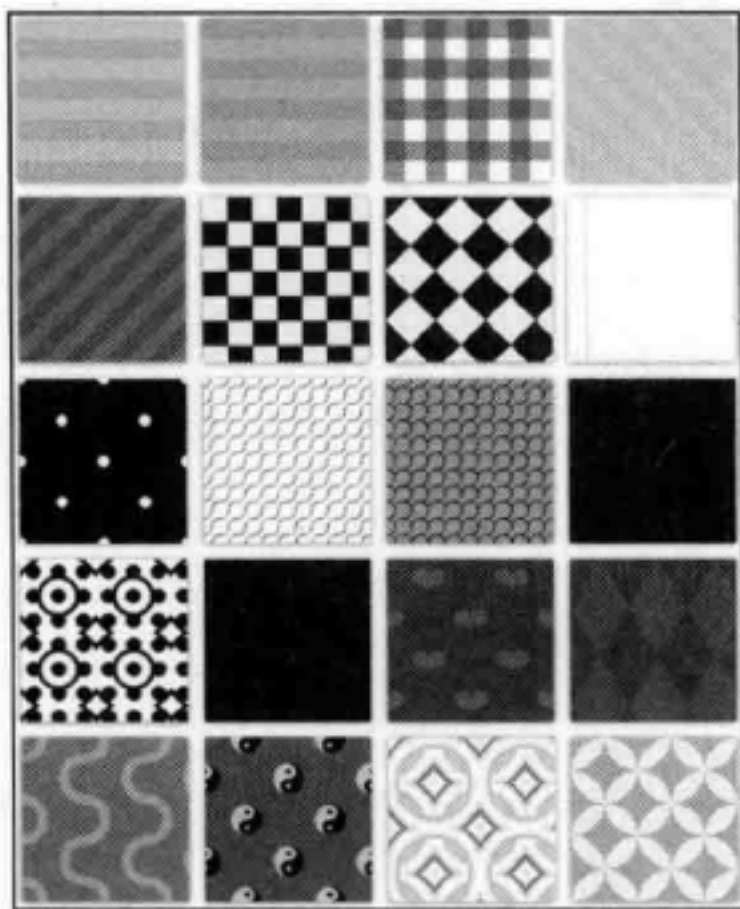
接下来利用前面所学的知识,并根据 Lea Verou[⊖]通过 CSS3 渐变属性制作的纹理图像为基础,在此演示 CSS3 渐变属性制作一些纹理图案。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>CSS3 渐变制作纹理图案 </title>
  <style type="text/css" media="screen">
    .patterns {
      width: 200px;
      height: 200px;
      float: left;
      margin: 10px;
      box-shadow: 0 1px 8px #666;
    }
    .pt1 {
      background-size: 50px 50px;
      background-color: #0ae;
      background-image: linear-gradient(rgba(255, 255, 255, .2) 50%,
        transparent 50%, transparent);
    }

    .pt2 {
      background-size: 50px 50px;
      background-color: #f90;
      background-image: linear-gradient(0deg, rgba(255, 255, 255, .2) 50%,
        transparent 50%, transparent);
    }
    ...// 省略部分代码
    .pt20 {
      background-color: #DDEEFF;
      background-image:
        radial-gradient(closest-side, transparent 98%, rgba(0, 0, 0, 0.3) 99%),
        radial-gradient(closest-side, transparent 98%, rgba(0, 0, 0, 0.3) 99%);
      background-position: 0 0px, 40px 40px;
      background-size: 80px 80px;
    }
  </style>
</head>
<body>
  <div class="patterns pt1"></div>
  <div class="patterns pt2"></div>
  ...
  <div class="patterns pt20"></div>
</body>
</html>
```

⊖ 详细内容见 <http://lea.verou.me/css3patterns>。

效果如图 10-56 所示。



☆图 10-56 CSS3 渐变制作纹理图案

10.6 本章小结

本章中主要介绍了 CSS3 的渐变属性，以及各种渐变的基础使用，以及浏览器兼容的处理方案。在 CSS3 中渐变特性主要包括线性渐变（linear-gradient）、径向渐变（radial-gradient）、重复线性渐变（repeating-linear-gradient）和重复径向渐变（repeating-radial-gradient）四种。其中线性渐变和重复线性渐变语法相同，而径向渐变和重复径向渐变的语法相同。

从本质上说，渐变就是背景属性中的 background-image，浏览器直接将渐变生成图片，可以直接使用 CSS 背景属性来控制它们。

CSS3 变形

CSS2.1 中的页面都是静态的，网页设计师也习惯把它作为页面效果的设计工具。

多年来，Web 设计师依赖于图片、Flash 或 JavaScript 才能完成修改页面的外观。CSS3 将改变设计师这种思维，借助 CSS3 可以轻松倾斜、缩放、移动以及翻转元素。下面将结合具体应用案例，详细讲解 CSS3 变形功能。

11.1 CSS3 变形简介

2012 年 9 月，W3C 组织发布了 CSS3 变形工作草案。允许 CSS 把元素转变为 2D 或 3D 空间，这个草案包括了 CSS3 2D 变形和 CSS3 3D 变形。

CSS3 变形是一些效果的集合，比如平移、旋转、缩放和倾斜效果，每个效果都称为变形函数（Transform Function），它们可以操控元素发生旋转、缩放、平移等变化。这些效果在之前都需要依赖图片、Flash 或 JavaScript 才能完成。而使用纯 CSS 来完成这些变形无须加载这些额外的文件，再一次提升了开发效率，提高了页面的执行效率。

11.1.1 CSS 变形属性及函数

CSS 变形允许动态的控制元素，可以在屏幕周围移动它们，缩小或扩大、旋转，或结合所有这些产生复杂的动画效果。通过 CSS 变形，可以让元素生成静态视觉效果，也可以很容易结合 CSS3 的 transition 和动画的 keyframe 产生一些动画效果。

□ CSS3 变形中具有 X /Y 可用的函数：translateX()、translateY()、scaleX()、scaleY()、skewX() 和 skewY()。






- CSS3 2D 变形函数包括：translate()、scale()、rotate() 和 skew()。translate() 函数接受 CSS 的标准度量单位；scale() 函数接受一个 0 ~ 1 之间的十进制值；rotate() 和 skew() 两个函数都接受一个径向的度量单位值 deg。除了 rotate() 函数之外，每个函数都接受 X 轴和 Y 轴的参数。2D 变形中还有一个矩阵 matrix() 函数，包括 6 个参数。
- CSS3 3D 变形函数包括：rotateX()、rotateY()、rotate3d()、translateZ()、translate3d()、scaleZ() 和 scale3d()。3D 变形中也包括一个矩阵 matrix3d() 函数，包括 16 个参数。

11.1.2 浏览器兼容性

1. 2D 变形兼容性

目前为止 CSS3 的 2D 变形在主流浏览器中得到较好的支持，如表 11-1 所示。

表 11-1 2D 变形兼容性

属性名称					
2D transform	9+ ✓	4.0+ ✓	3.5+ ✓	3.1 ✓	10.5+ ✓






CSS3 的 2D 变换虽然得到众多主流浏览器的支持，但在实际使用的时候需要添加浏览器各自的私有属性。

- IE 9 中使用 2D 变形时，需要添加 -ms- 私有属性，在 IE 10+ 版本开始支持标准版本。
- Firefox 3.5 至 Firefox 15.0 版本需要添加 -moz- 的私有属性，在 Firefox 16+ 版本开始支持标准版本。
- Chrome 4.0+ 开始支持 2D 变形，在实际使用的时候需要添加 -webkit- 私有属性。
- Safari 3.1+ 开始支持 2D 变形，在实际使用的时候需要添加 -webkit- 私有属性。
- Opera 10.5+ 开始支持 2D 变形，在实际使用的时候需要添加 -o- 私有属性，但在 Opera 12.1 版本不需要添加私有属性，不过在 Opera 15.0+ 版本需要添加私有属性 -webkit- 私有属性。
- 移动设备 iOS Safari 3.2+、Android Browser 2.1+、Blackberry Browser 7.0+、Opera Mobile 14.0+、Chrome for Android 25.0+ 需要添加私有属性 -webkit-，而 Opera Mobile 11.0 至 Opera Mobile 12.1 和 Firefox for Android 19.0+ 不需要使用浏览器私有属性。

2. 3D 变形兼容性

3D 变形也得到了众多主流浏览器的支持，只不过比 2D 变形来得晚些，详细请看表 11-2。

表 11-2 3D 变形浏览器兼容性

属性名称					
3D 变形	10+ ✓	10.0+ ✓	12.0+ ✓	4.0+ ✓	15.0+ ✓

3D 变形在实际使用这时同样需要添加各浏览器的私有属性，并且有个别属性在某些主流浏览器中并未得到很好的支持。

❑ IE 10+ 中 3D 变形部分属性未得到很好的支持。

❑ Firefox 10.0 至 Firefox 15.0 版本的浏览器，在使用 3D 变形时需要添加私有属性 `-moz-`，但从 Firefox 16.0+ 版本开始无需添加浏览器私有属性。

❑ Chrome 12.0+ 版本中使用 3D 变形时需要添加私有属性 `-webkit-`。

❑ Safari 4.0+ 版本中使用 3D 变形时需要添加私有属性 `-webkit-`。

❑ Opera 15.0+ 版本才开始支持 3D 变形，使用时需要添加私有属性 `-webkit-`。

❑ 移动设备中 iOS Safari 3.2+、Android Browser 3.0+、Blackberry Browser 7.0+、Opera Mobile 14.0+、Chrome for Android 25.0+ 都支持 3D 变形，但在使用时需要添加私有属性 `-webkit-`；Firefox for Android 19.0+ 支持 3D 变形，但无须添加浏览器私有属性。

11.2 CSS 变形属性详解

`transform` 属性指一组转换函数，`transform-origin` 属性指定元素的中心点在哪，新增增加了第三个数 `transform-origin-z`，控制元素三维空间中心点。`transform-style` 的值设置为 `preserve-3d`，建立一个 3D 渲染环境。

11.2.1 transform 属性

`transform` 属性让元素在一个坐标系统中变形，包含一系列变形函数，可以移动、旋转和缩放元素。

1. transform 属性的语法

`transform` 属性的基本语法如下。

```
transform: none | <transform-function> [<transform-function>]*
```

可用于内联元素和块元素。其默认值为 `none`，表示元素不进行变形。另一个属性值是一系列的 `<transform-function>`，表示一个或多个变形函数，以空格分开；换句话说就是同时对一个元素进行变形的多种属性操作，例如 `rotate`、`scale`、`translate` 等。这里需要提醒大家，以往叠加效果都是用逗号（,）隔开，但在 `transform` 中使用多个 `transform-function` 时却需要用空格隔开。

2. transform-function 介绍

所有的 2D 变形函数也包含于 3D 变形规范中。如此一来，CSS3 变形中的函数根据不同的规范略有不同，下面列出的是变形中的 2D 和 3D 常用变形函数的功能，如表 11-3 和表 11-4 所示。

表 11-3 2D transform 常用的 transform-function 的功能

函数	功能描述
translate()	移动元素，可以根据 X 轴和 Y 轴坐标重新定位元素位置。在此基础上有两个扩展函数 translateX() 和 translateY()
scale()	缩小或放大元素，可以使用元素尺寸发生变化。在此基础上有两个扩展函数 scaleX() 和 scaleY()
rotate()	旋转元素
skew()	让元素倾斜。在此基础上有两个扩展函数 skewX() 和 skewY()
matrix()	定义矩阵变形，基于 X 轴和 Y 轴坐标重新定位元素位置

表 11-4 3D transform 常用的 transform-function 的功能

函数	功能描述
translate3d()	移动元素，用来指定一个 3D 变形移动位移量
translate()	指定 3 D 位移在 Z 轴的位移量
scale3d()	缩放一个元素
scaleZ()	指定 Z 轴的缩放向量
rotate3d()	指定元素具有一个三维旋转的角度
rotateX()、rotateY()、rotateZ()	让元素具有一个旋转角度
perspective()	指定一个透视投影矩阵
matrix3d()	定义矩阵变形

3. 实战体验：疯狂的按键

这个示例制作一系列的分享按钮，当鼠标悬停在每个按钮上，按钮会旋转并且同时放大，鼠标一旦移开，按钮会回到默认状态。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>CSS3 制作旋转 icon</title>
  <style type="text/css">
/* 基本样式省略 */
/* 设置动画效果 */
#socialicons img,
#socialicons2 img{
  transition: all 0.8s ease-in-out;
}
/* 设置变形，顺时针旋转 360 度，放大 1.5 倍 */
#socialicons img:hover{
  transform: rotate(360deg) scale(1.5);
}
/* 设置变形，逆时针旋转 360 度，缩小 0.7 倍 */
#socialicons2 img:hover{
  transform: rotate(-360deg) scale(.7);
}
```



```

</style>
</head>
<body>
<h1> 顺时针旋转 360 度放大 1.2 倍 </h1>
<ul id="socialicons">
<li><a href="#"></a></li>
<li><a href="#"></a></li>
<li><a href="#"></a></li>
<li><a href="#"></a></li>
<li><a href="#"></a></li>
</ul>
<h1> 逆时针旋转 360 度缩小 0.7 倍 </h1>
<ul id="socialicons2">
<li><a href="#"></a></li>
<li><a href="#"></a></li>
<li><a href="#"></a></li>
<li><a href="#"></a></li>
<li><a href="#"></a></li>
</ul>
</body>
</html>

```

第一个列表中，鼠标悬停在按钮上时，对每个图片设置顺时针旋转 360 度、放大 1.5 倍。

```

#socialicons img:hover{
    transform: rotate(360deg) scale(1.5);
}

```

第二个列表中，鼠标悬停在按钮上时，对每个图片设置逆时针旋转 360 度、缩小 0.7 倍。

```

#socialicons2 img:hover{
    transform: rotate(-360deg) scale(.7);
}

```

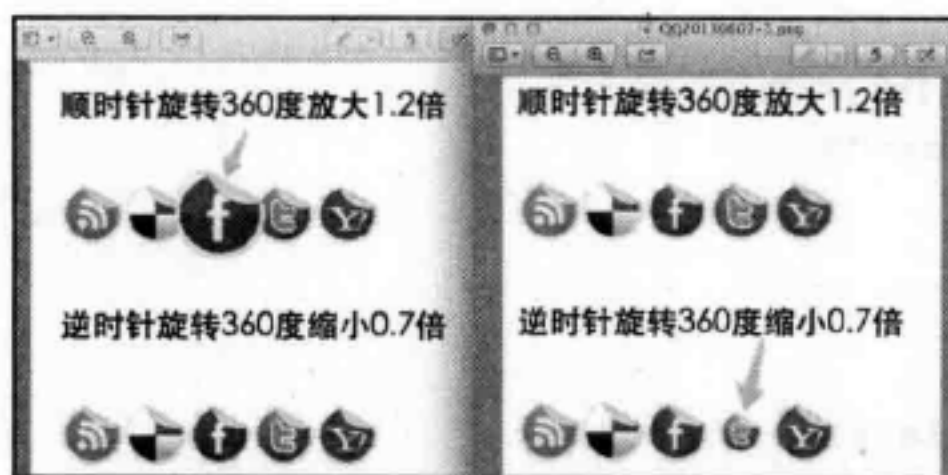


图 11-1 不断旋转和缩放效果

效果如图 11-1 所示。



注意

为了节约文章篇幅，下面代码示例中都省略了各浏览器前缀，在实际应用中需要添加各浏览器的私有前缀。

11.2.2 transform-origin 属性

transform-origin 属性用来指定元素的中心点位置。默认情况,变形的原点在元素的中心点,或者是元素 X 轴和 Y 轴的 50% 处,如图 11-2 所示。

没有使用 transform-origin 改变元素原点位置的情况下,CSS 变形进行的旋转、移位、缩放等操作都是以元素自己中心(变形原点)位置进行变形的。但很多时候需要在不同的位置对元素进行变形操作,这时就可以使用 transform-origin 来对元素进行原点位置改变,使元素原点不在元素的中心位置,以达到需要的原点位置。

如果把元素变换原点(transform-origin)设置 0(x)0(y),元素的变换原点转换到元素的左顶角处,如图 11-3 所示。

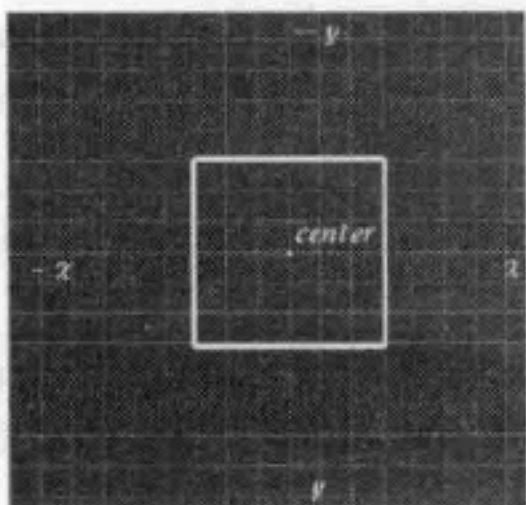


图 11-2 元素的变形原点

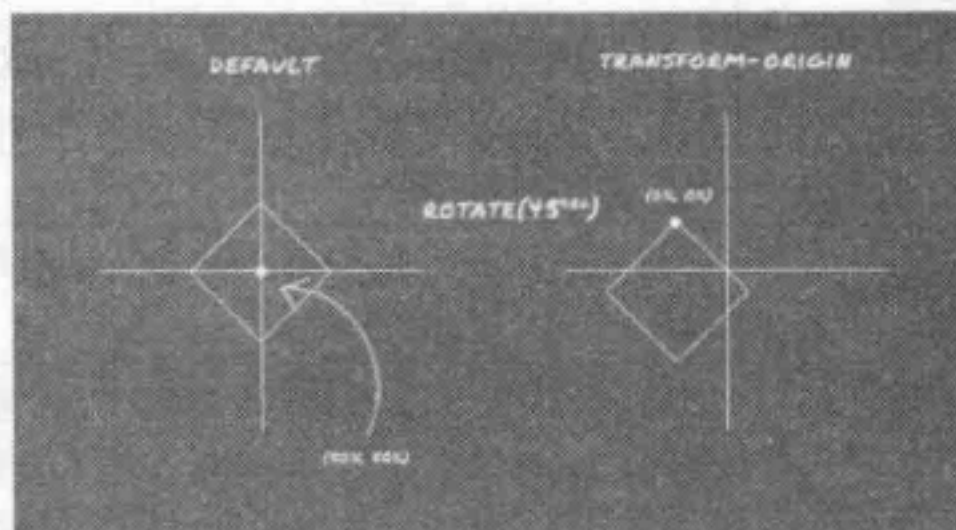


图 11-3 把元素的变形原点从中心点重置为元素左顶角

1. transform-origin 属性的语法

如图 11-3 所示,改变 transform-origin 属性的 X 轴和 Y 轴的值就可以重置元素变形原点位置,其基本语法如下所示。

```
transform-origin: [<percentage> | <length> | left | center | right | top | bottom] |
[<percentage> | <length> | left | center | right] | [[<percentage> | <length> | left | center
| right] && [<percentage> | <length> | top | center | bottom]] <length> ?
```

上面的语法让人看得发晕,其实可以将语法拆分成以下形式。

/* 只设置一个值的语法 */

```
transform-origin: x-offset
```

```
transform-origin: offset-keyword
```

/* 设置两个值的语法 */

```
transform-origin: x-offset y-offset
```

```
transform-origin: y-offset x-offset-keyword
```

```
transform-origin: x-offset-keyword y-offset
```

```
transform-origin: x-offset-keyword y-offset-keyword
```

```
transform-origin: y-offset-keyword x-offset-keyword
```

/* 设置三个值的语法 */

```
transform-origin: x-offset y-offset z-offset
```

```
transform-origin: y-offset x-offset-keyword z-offset
```

```
transform-origin: x-offset-keyword y-offset z-offset
```

```
transform-origin: x-offset-keyword y-offset-keyword z-offset  
transform-origin: y-offset-keyword x-offset-keyword z-offset
```

transform-origin 属性值可以是百分比、em、px 等具体的值，也可以是 top、right、bottom、left 和 center 这样的关键词。

2D 变形中的 transform-origin 属性可以是一个参数值，也可以是两个参数值。如果是两个参数值时，第一值设置水平方向 X 轴的位置，第二个值是用来设置垂直方向 Y 轴的位置。

3D 变形中的 transform-origin 属性还包括了 Z 轴的第三个值，其各个值的取值简单说明如表 11-5 所示。

表 11-5 3D 变形中的 transform-origin 属性

属性值	功能描述
x-offset	用来设置 transform-origin 水平方向 X 轴的偏移量，可以使用 <length> 和 <percentage> 值，同时可以是正值（从中心点沿水平方向 X 轴向右偏移量），也可以是负值（从中心点沿水平方向 X 轴向左偏移量）
offset-keyword	是 top、right、bottom、left 或 center 中的一个关键词，可以用来设置 transform-origin 的偏移量
y-offset	用来设置 transform-origin 属性在垂直方向 Y 轴的偏移量，可以使用 <length> 和 <percentage> 值，同时可以是正值（从中心点沿垂直方向 Y 轴向下的偏移量），也可以是负值（从中心点沿垂直方向 Y 轴向上的偏移量）
x-offset-keyword	是 left、right 或 center 中的一个关键词，可以用来设置 transform-origin 属性值在水平 X 轴的偏移量
y-offset-keyword	是 top、bottom 或 center 中的一个关键词，可以用来设置 transform-origin 属性值在垂直方向 Y 轴的偏移量
z-offset	设置 3D 变形中 transform-origin 远离用户眼睛视点的距离，默认值 z=0，其取值可以是 <length>，不过 <percentage> 在这里将无效

看上去 transform-origin 取值与 background-position 取值类似。为了方便记忆，可以把关键词和百分比值对比来记，如表 11-6 所示。

表 11-6 transform-origin 取值为关键词与百分比的对比

关键词	百分比
top = top center = center top	50% 0%
right = right center = center right	100% 或 (100% 50%)
bottom = bottom center = center bottom	50% 100%
left = left center = center left	0% 或 (0% 50%)
center = center center	50% 或 (50% 50%)
top left = left top	0% 0%
right top = top right	100% 0%
bottom right = right bottom	100% 100%
bottom left = left bottom	0% 100%

为了让大家能一眼看明白，以 transform 中的旋转 rotate() 为例，transform-origin 取值不一样时的效果，如表 11-7 所示。

表 11-7 transform-origin 取值及图示

取值	图示	取值	图示	取值	图示
center center center 50% 50% 50%		bottom bottom center center bottom 50% 100%		right top top right 100% 0	
top top center center top 50% 0		left left center center left 0 0 50%		bottom right right bottom 100% 100%	
right right center center right 100% 100% 50%		top left left top 0 0		left bottom bottom left 0 100%	

CSS3 变形中旋转、缩放、倾斜都可以通过 transform-origin 属性重置元素的原点，但其中的位移 translate() 始终以元素中心点进行位移。例如下面的两段代码的演示过程。

```
div {
  transform-origin: 50% 50%;
  transform: translate(40px, 40px) translate(-50px, 35px) translateY(30px);
}
```

接下来通过 transform-origin 将变形原点设置为 100% 100%。

```
div {
  transform-origin: 100% 100%;
  transform: translate(40px, 40px) translate(-50px, 35px) translateY(30px);
}
```

虽然元素的变形原点通过 transform-origin 从 50% 50% 变成 100% 100%，但元素位移 translate() 始终是依元素中心点进行位移，如图 11-14 所示。

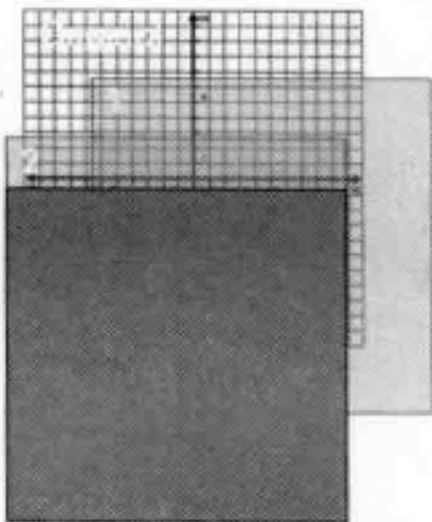



图 11-4 transform-origin 设置为 50% 50% 和 100% 100% 的 translate() 效果

到写本节内容时，transform-origin 属性在现代主流浏览器得到很好的支持，但在一些浏览器之下依然需要添加各浏览器私有属性，详细情况如表 11-8 所示。

表 11-8 transform-origin 属性浏览器兼容性

transform-origin	需要添加浏览器私有属性版本	支持 W3C 标准规范的浏览器
2D 变形	IE 9+、Firefox 3.5+、Chrome 4+、Safari 3.1+、Opera 10.5+、iOS Safari 3.2+、Android Browser 2.1+、Blackberry Browser 7.0+、Chrome for Android 25.0+	IE10+、Firefox 16+、Opera 12.1+、Opera Mobile 11.0+、Firefox for Android 19.0
3D 变形	IE 10+、Firefox 10+、Chrome 12+、Safari 4+、Opera 15+、iOS Safari 3.2+、Android Browser 3.0+、Blackberry Browser 7.0+、Opera Mobile 14.0+、Chrome for Android 25.0+	Firefox16+、Firefox for Android 19+

 **注意** Opera 15.0+ 和 Opera Mobile 14.0+ 开始需要添加私有前缀 -webkit-。

通过 transform-origin 属性改变元素的原点，可以实现不同的变形效果，下面的示例中分别演示了改变元素原点前后，CSS3 变形各函数对图像变形操作。

2. 实战体验：改变对象原点

首先来看 transform-origin 属性改变元素原点前后，rotate() 函数对图像的旋转效果。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>transform-origin 改变元素原点前后 rotate() 旋转图像效果 </title>
  <style type="text/css">
    /* 基本样式省略 */
    div img:nth-child(1){
      opacity: .5;
      z-index: 1;
      transform: rotate(10deg);
    }
    div img:nth-child(2){
      opacity: .6;
      z-index: 2;
      transform: rotate(25deg);
    }
    div img:nth-child(3){
      opacity: .7;
      z-index: 3;
      transform: rotate(35deg);
    }
    div img:nth-child(4){
      opacity: .8;
      z-index: 4;
      transform: rotate(45deg);
    }
  </style>
</head>
<body>
  <div>
    <img alt="Image 1" data-bbox="141 516 200 540"/>
    <img alt="Image 2" data-bbox="210 516 270 540"/>
    <img alt="Image 3" data-bbox="280 516 340 540"/>
    <img alt="Image 4" data-bbox="350 516 410 540"/>
  </div>
</body>
</html>
```

```

}
div img:nth-child(5){
    z-index: 5;
    transform: rotate(60deg);
}
div:nth-of-type(2) img {
    transform-origin: bottom;
}

```

上面实例演示了变形中旋转 `rotate()` 函数围绕不同原点旋转的效果, 第一个容器 `div` 中的图片围绕图片默认原点 (中心) 旋转的过程; 而第二个容器 `div` 中的图片经过 `transform-origin` 属性将图片原点从中心点 (`center`) 修改为底部中心点 (`bottom`) 旋转过程。效果如图 11-5 所示。

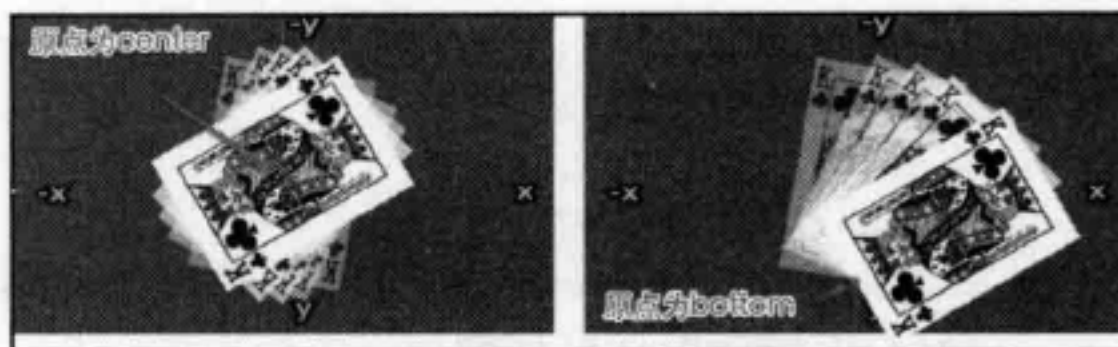


图 11-5 `transform-origin` 修改图像原点旋转效果

接下来, `transform-origin` 修改原点前后, CSS3 变形中倾斜 `skew()` 函数对图片变形的过程。

```

<!DOCTYPE HTML>
<html lang="en-US">
<head>
    <meta charset="UTF-8">
    <title>transform-origin 改变元素原点前后 skew() 倾斜图像效果 </title>
    <style type="text/css">
        /* 基本样式省略 */
        div img:nth-child(1){
            opacity: .5;
            z-index: 1;
            transform: skewX(10deg);
        }
        div img:nth-child(2){
            opacity: .6;
            z-index: 2;
            transform: skewX(15deg);
        }
        div img:nth-child(3){
            opacity: .7;
            z-index: 3;
            transform: skewX(20deg);
        }
        div img:nth-child(4){
            opacity: .8;
            z-index: 4;
            transform: skewX(25deg);
        }
        div img:nth-child(5){
            z-index: 5;

```



```

    transform: skewX(30deg);
  }
  div:nth-of-type(2) img {
    transform-origin: bottom;
  }
  ...

```

效果如图 11-6 所示。

变形中的缩放 `scale()` 函数在不同原点产生变形效果。

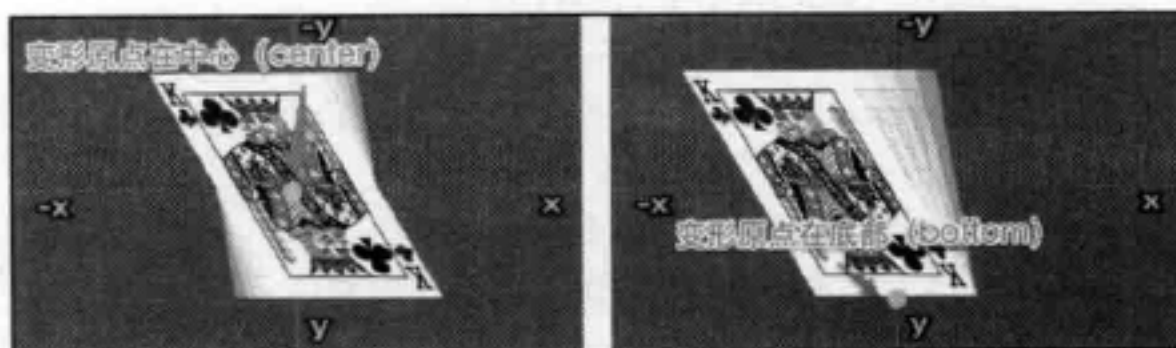


图 11-6 `transform-origin` 修改图像原点后倾斜 `skewX()` 效果

```

<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>transform-origin 改变元素原点前后 scale() 缩放图像效果</title>
  <style type="text/css">
    /* 基本样式省略 */
    div img:nth-child(1){
      opacity: .5;
      z-index: 1;
      transform: scale(1.2);
    }
    div img:nth-child(2){
      opacity: .6;
      z-index: 2;
      transform: scale(1.1);
    }
    div img:nth-child(3){
      opacity: .7;
      z-index: 3;
      transform: scale(.9);
    }
    div img:nth-child(4){
      opacity: .8;
      z-index: 4;
      transform: scale(.8);
    }
    div img:nth-child(5){
      z-index: 5;
      transform: scale(.6);
    }
    div:nth-of-type(2) img {
      transform-origin: right;
    }
    ...

```

效果如图 11-7 所示。

上面三个简单实例再次验证了 CSS3 变形中的旋转 `rotate()`、缩放 `scale()` 和倾斜 `skew()`

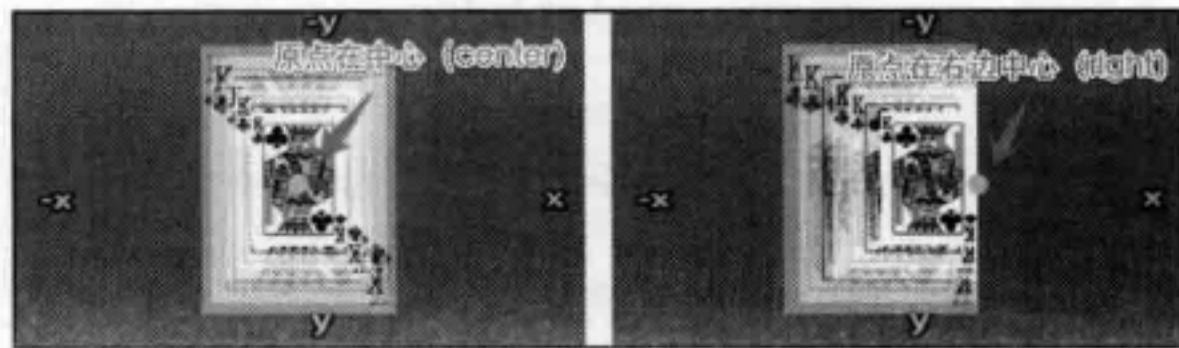


图 11-7 `transform-origin` 修改图像原点后缩放 `scale()` 效果

函数都可以通过 transform-origin 属性来改变元素对象的原点位置。但是 transform-origin 属性改变元素对象原点位置，位移 translate() 函数始终会根据元素对象中心点进行位移。

前面演示的只是 2D 变形中 transform-origin 用来修改元素对象原点，以及对各种变形函数产生的不同效果。接下来，用一个简单的实例演示 3D 变形中 transform-origin 修改元素原点的 3D 旋转效果。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>transform-origin 改变元素原点前后 rotate3d() 旋转图像效果</title>
  <style type="text/css">
    div {
      width: 500px;
      height: 300px;
      margin: 30px auto;
      position: relative;
      background: url(images/bg-grid.jpg) no-repeat center center;
      background-size: 100% 100%;
    }
    div img {
      position: absolute;
      top: 50%;
      left: 50%;
      margin-left: -71px;
      margin-top: -100px;
      backface-visibility: visible;
      transform: perspective(500px);
    }
    div img:nth-child(1){
      opacity: .5;
      z-index: 1;
      transform: rotate3d(1, 1, 1,10deg);
    }
    div img:nth-child(2){
      opacity: .6;
      z-index: 2;
      transform: rotate3d(1, 1, 1,25deg);
    }
    div img:nth-child(3){
      opacity: .7;
      z-index: 3;
      transform: rotate3d(1, 1, 1,35deg);
    }
    div img:nth-child(4){
      opacity: .8;
      z-index: 4;
      transform: rotate3d(1, 1, 1,45deg);
    }
  </style>
</head>
<body>
  <div>
    <img alt="A grid background with four semi-transparent images rotated 10, 25, 35, and 45 degrees in 3D space." data-bbox="125 234 933 917"/>
  </div>
</body>
</html>
```

```

    }
    div img:nth-child(5){
      z-index: 5;
      transform: rotate3d(1, 1, 1,60deg);
    }
    div:nth-of-type(2) img {
      transform-origin: left bottom -50px;
    }
  }
  ...

```

效果如图 11-8 所示。

11.2.3 transform-style 属性

transform-style 属性是 3D 空间一个重要属性，指定嵌套元素如何在 3D 空间中呈现。主要有两个属性值：flat 和 preserve-3d。

1. transform-style 属性的语法

transform-style 属性的使用语法非常简单。

```
transform-style: flat | preserve-3d
```

❑ flat: 默认值，表示所有子元素在 2D 平面呈现。

❑ preserve-3d: 所有子元素在 3D 空间中呈现。如果对一个元素设置了 transform-style 的值为 flat，该元素的所有子元素都将被平展到该元素的 2D 平面中进行呈现。沿着 X 轴或 Y 轴方向旋转该元素将导致位于正或负 Z 轴位置的子元素显示在该元素的平面上，而不是它的前面或者后面。

如果对一个元素设置了 transform-style 的值为 preserve-3d，它表示不执行平展操作，它的所有子元素位于 3D 空间中。

2. 实战体验：纸牌翻转

transform-style 属性需要设置在父元素中，并且高于任何嵌套的变形元素。最后，运用一个翻转的例子，加深一下对 transform-style 属性的印象。

```

<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>transform-style 的使用 </title>
  <style type="text/css">
    .wrap {
      width: 500px;
      height: 300px;
      margin: 30px auto;
      position: relative;

```

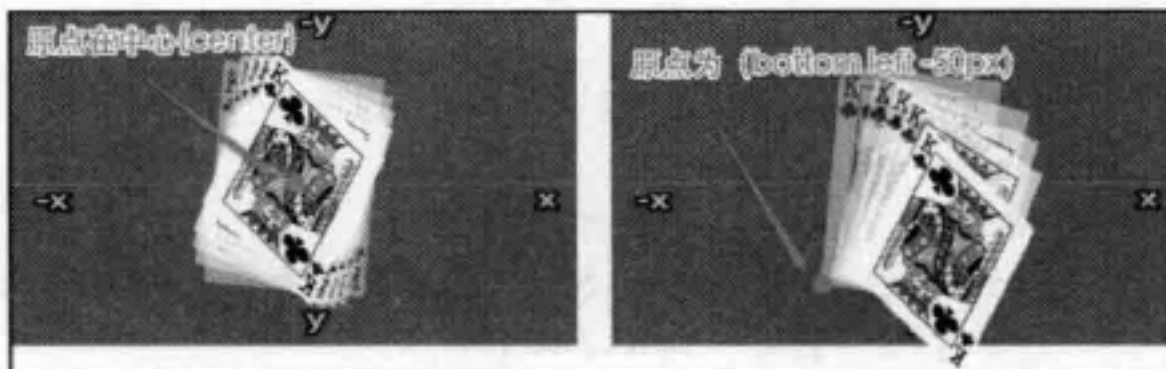


图 11-8 3D 变形中 transform-origin 修改

图像原点前后 3D 旋转效果


```

background: url(images/bg-grid.jpg) no-repeat center center;
background-size: 100% 100%;
}
/* 设置动画 */

@keyframes spin{
  0%{
    transform: rotateY(0deg)
  }
  100%{
    transform: rotateY(360deg)
  }
}

.spin {
  width: 142px;
  height: 200px;
  position: absolute;
  top: 50%;
  left: 50%;
  margin-left: -72px;
  margin-top: -101px;
  border: 1px dashed orange;
  cursor: pointer;
  transform-style: preserve-3d;
}
/* 调用动画 */
.spin:hover{
  animation: spin 5s linear infinite;
}

.rotate {
  background: url(images/cardKingClub.png) no-repeat center;
  background-size: 100% 100%;
  border: 5px solid hsla(50,50%,50%,.9);
  transform: perspective(200px) rotateY(45deg);
}

.rotate img{
  border: 1px solid green;
  transform: rotateX(15deg) translateZ(10px);
  transform-origin: 0 0 40px;
}
/* 改变 transform-style 的默认值 */
.three-d {
  transform-style: preserve-3d;
}
...

```

效果如图 11-9 所示。

正如所看到的，元素设置 `.rotate` 设置了 `flat` 值时，其子元素 `img` 的不会根据 `translateZ()` 值摊开，而在同一平面旋转，如图 11-9 所

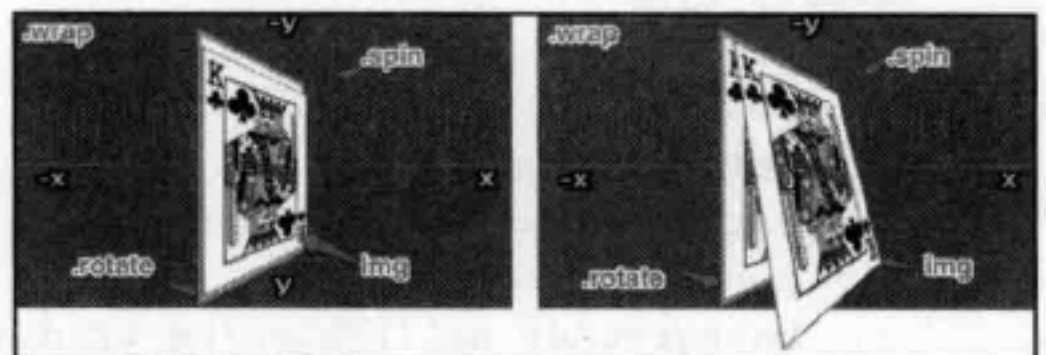


图 11-9 transform-style 设置 flat (左) 和 preserve-3d (右) 的旋转位移效果

示；元素 `.rotate` 设置了 `preserve-3d` 值时，其子元素 `img` 会根据 `translateZ()` 值摊开，不再会堆叠在一起。

有一点需要特别提醒，如果元素设置了 `transform-style` 值为 `preserve-3d`，就不能为了防止子元素溢出容器而设置 `overflow` 值为 `hidden`，如果设置了 `overflow:hidden` 同样可以迫使子元素出现在同一平面（和元素设置了 `transform-style` 为 `flat` 一样的效果），如图 11-10 所示。

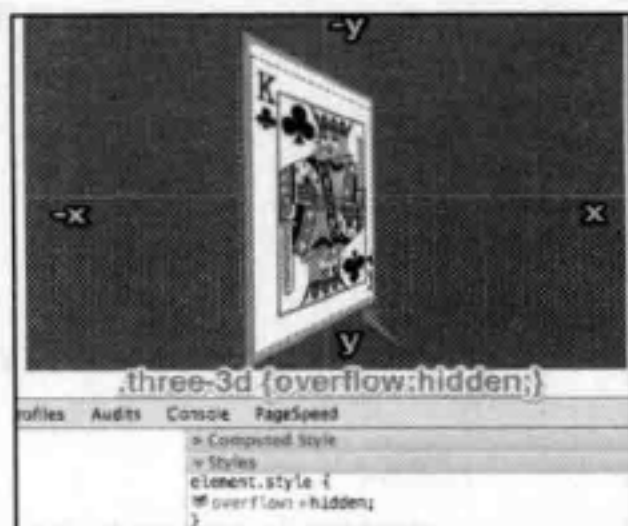


图 11-10 `transform-style` 设置 `preserve-3d` 和 `overflow:hidden` 值的效果

11.2.4 perspective 属性

`perspective` 属性对于 3D 变形来说至关重要，该属性会设置查看者的位置，并将可视内容映射到一个视锥上，继而投到一个 2D 视平面上。如果不指定透视，则 Z 轴空间中的所有点将平铺到同一个 2D 视平面中，并且变换结果中将不存在景深概念。

上面的描述可能让人难以理解一些，其实对于 `perspective` 属性，可以简单地理解为视距，用来设置用户和元素 3D 空间 Z 平面之间的距离。而其效应由他的值来决定，值越小，用户与 3D 空间 Z 平面距离越近，视觉效果更令人印象深刻；反之，值越大，用户与 3D 空间 Z 平面距离越远，视觉效果就很小。

在 3D 变形中，对于某些变形（例如下面的示例演示的沿 Z 轴的变形），`perspective` 属性对于查看变形的效果来说必不可少。

1. 实战体验：纸牌 3D 旋转

制作一个扑克牌 3D 旋转效果，并且一个在扑克牌的父元素添加了视距 `perspective`，而另一个却没有设置。

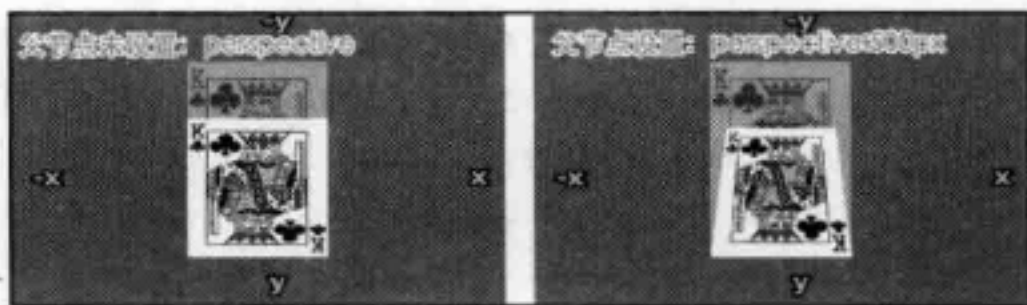
```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>perspective 的使用 </title>
  <style type="text/css">
    div {
      width: 500px;
      height: 300px;
      margin: 30px auto;
      position: relative;
      background: url(images/bg-grid.jpg) no-repeat center center;
      background-size: 100% 100%;
    }
    div img {
      position: absolute;
      top: 50%;
```

```

    left: 50%;
    margin-left: -71px;
    margin-top: -100px;
    transform-origin: bottom;
  }
  div img:nth-child(1){
    opacity: .5;
    z-index: 1;
  }
  div img:nth-child(2){
    z-index: 2;
    transform: rotateX(45deg);
  }

  div:nth-of-type(2){
    perspective: 500px;
  }
  ...

```



效果如图 11-11 所示。

图 11-11 父节点设置 perspective 前后的 rotateX() 效果

父节点没设置 perspective 的情况下, 梅花 K 的 3D 旋转效果不明显, 而在父节点设置 perspective 后, 梅花 K 的 3D 旋转很明显。

2. perspective 属性的语法

上例简单演示了 perspective 的使用方法, 回过头来, 看看 perspective 的使用语法。

perspective: none | <length>

perspective 包括两个属性。

- ❑ none: 默认值, 表示无限的角度来看 3D 物体, 但看上去是平的。
- ❑ <length>: 接受一个长度单位大于 0 的值, 其单位不能为百分比值。值越大, 角度出现的越远, 从而创建一个相当低的强度和非常小的 3D 空间变化。反之, 值越小, 角度出现的越近, 从而创建一个高强度的角度和一个大型 3D 变化。

比如站在 10 千米和 1000 千米的地方看一个 10 千米的立方体。在 10 千米的地方, 距离立方体是一样的尺寸。因此视角转变远远大于站在 1000 千米处的, 立体尺寸是距离立方体距离的百分之一。同样的思维适用于 perspective 的 <length> 值。一起来看一个实例, 来加强这方面的理解。

```

<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>perspective 的使用 </title>
  <style type="text/css">
    .wrapper {
      width: 50%;

```



```

    float: left;
}
.cube {
  font-size: 4em;
  width: 2em;
  margin: 1.5em auto;
  transform-style: preserve-3d;
  transform: rotateX(-40deg) rotateY(32deg);
}
.side {
  position: absolute;
  width: 2em;
  height: 2em;
  background: rgba(255, 99, 71, 0.6);
  border: 1px solid rgba(0, 0, 0, 0.5);
  color: white;
  text-align: center;
  line-height: 2em;
}
.front {
  transform: translateZ(1em);
}
.top {
  transform: rotateX(90deg) translateZ(1em);
}
.right {
  transform: rotateY(90deg) translateZ(1em);
}
.left {
  transform: rotateY(-90deg) translateZ(1em);
}
.bottom {
  transform: rotateX(-90deg) translateZ(1em);
}
.back {
  transform: rotateY(-180deg) translateZ(1em);
}
.w1 {
  perspective: 100px;
}
.w2 {
  perspective: 1000px;
}
</style>
</head>
<body>
<div class="wrapper w2">
  <div class="cube">
    <div class="side front">1</div>
    <div class="side back">6</div>

```

```

<div class="side right">4</div>
<div class="side left">3</div>
<div class="side top">5</div>
<div class="side bottom">2</div>
</div>
</div>
<div class="wrapper w1">
  <div class="cube">
    <div class="side front">1</div>
    <div class="side back">6</div>
    <div class="side right">4</div>
    <div class="side left">3</div>
    <div class="side top">5</div>
    <div class="side bottom">2</div>
  </div>
</div>
</body>
</html>

```

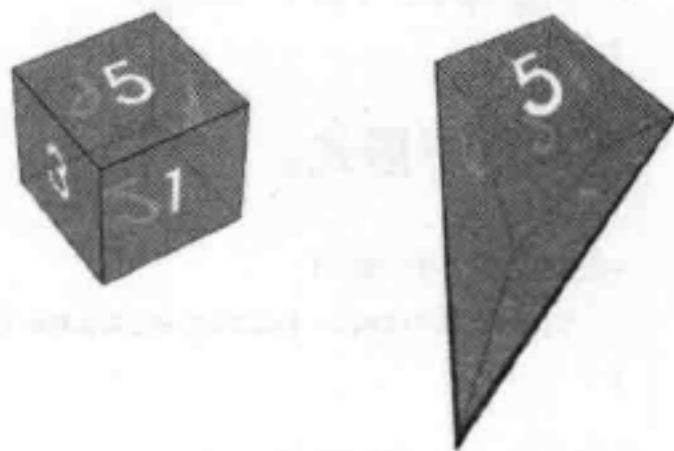


图 11-12 perspective 取值 100px 和 1000px 的 3D 立方体效果

效果如图 11-12 所示。

依据上面的介绍，可对 perspective 取值做一个简单的结论。

- perspective 取值为 none 或不设置，没有 3D 空间。
- perspective 取值越小，3D 效果就越明显，也就是眼睛越靠近真 3D。
- perspective 的值无穷大，或值为 0 时与取值为 none 效果一样。

为了更好地理解 perspective 属性，很有必要把它和 translateZ 的关系结合起来。其实也可以把 perspective 的值简单地理解为人的眼睛到显示器的距离，而 translate 就是 3D 物体距离源点的距离，下面引用 W3C 的一张图来解说 perspective 和 translateZ 的关系。perspective 属性和 translateZ 的位置比例如图 11-13 所示。

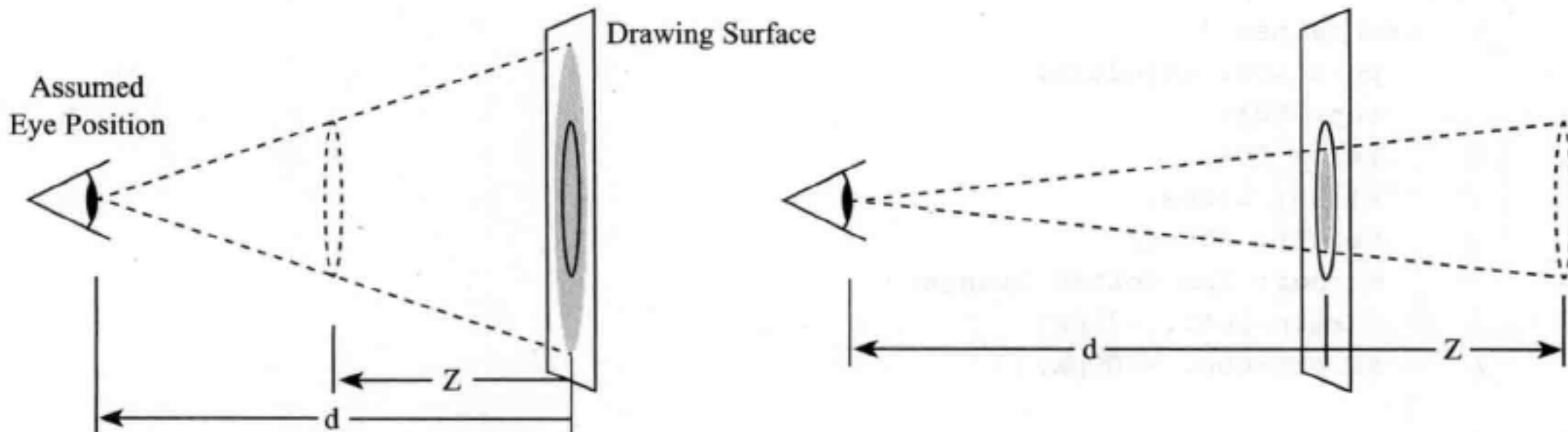


图 11-13 perspective 和 translateZ 的关系

Z 是半个 d，为了使用原始圆（轮廓）看起来出现在 Z 轴上（虚线圆），画布上的实体圆将扩大两部，如浅蓝色的圆。在底部图中显示，圆按比例缩小，致使虚线圆出现在画布后面，并且 Z 的大小是距原始位置三分之一。

3. perspective() 函数与 perspective 属性

在 3D 变形中，除了 perspective 属性可以激活一个 3D 空间之外，在 3D 变形的函数中的 perspective() 也可以激活 3D 空间。不同的地方是：perspective 用在舞台元素上（变形元素们的共同父元素）；perspective() 就是用在当前变形元素上，并且可以与其他的 transform 函数一起使用。例如，可以把：

```
.stage {
  perspective: 600px
}
```

写成以下形式。

```
.stage .box {
  transform: perspective(600px);
}
```

来看一个简单示例。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>perspective 和 perspective() 对比</title>
  <style type="text/css">
    .stage {
      width: 500px;
      height: 300px;
      margin: 30px auto;
      position: relative;
      background: url(images/bg-grid.jpg) no-repeat center center;
      background-size: 100% 100%;
    }
    .container {
      position: absolute;
      top: 50%;
      left: 50%;
      width: 142px;
      height: 200px;
      border: 1px dotted orange;
      margin-left: -71px;
      margin-top: -100px;
    }
    .container img{
      transform: rotateY(45deg);
    }
    .stage:nth-child(1) .container{
      perspective: 600px;
    }
    .stage:nth-child(2) img {
```



```

    transform:perspective(600px) rotateY(45deg);
  }
  ...

```

效果如图 11-14 所示。

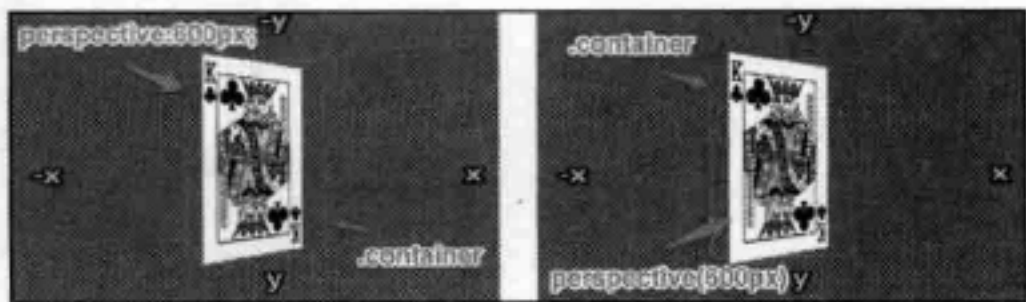


图 11-14 perspective 和 perspective() 的对比效果

图 11-14 效果可以看出，虽然书写的形式、属性名称不一致，但是效果却一样。

- ❑ perspective 属性和 perspective() 函数功能一样，但其取值以及运用的对象有所不同。
- ❑ perspective 属性可以取值为 none 或长度值；而 perspective() 函数取值只能大于 0，如果取值为 0 或比 0 小的值，将无法激活 3D 空间；
- ❑ perspective 属性用于变形对象父元素；而 perspective() 函数用于变形对象自身，并和 transform 其他函数一起使用。

11.2.5 perspective-origin 属性

perspective-origin 属性是 3D 变形中另一个重要属性，主要用来决定 perspective 属性的源点角度。它实际上设置了 X 轴和 Y 轴位置，在该位置观看者好像在观看该元素的子元素。

1. perspective-origin 属性的语法

perspective-origin 属性的使用语法也很简单。

```

perspective-origin: [<percentage> | <length> | left | center | right | top | bottom] |
    [[<percentage> | <length> | left | center | right] && [<percentage> |
    <length> | top | center | bottom]]

```

该属性默认值为“50% 50%”（也就是 center），可以设置为一个值，也可以设置为两个长度值。

第一个长度值指定相对于元素的包含框的 X 轴上的位置。它可以是长度值（以受支持的长度单位表示）、百分比或以下三个关键词之一。

- ❑ left：在包含框的 X 轴方向长度的 0%。
- ❑ center：中间点。
- ❑ right：长度的 100%。

第二个长度值指定相对于元素的包含框的 Y 轴上的位置。它可以是长度值、百分比或以下三个关键词之一。

- ❑ top: 在包含框的 Y 轴方向长度的 0%。
- ❑ center: 中间点。
- ❑ bottom: 长度的 100%。

注意，为了指转换子元素变形的深度，`perspective-origin` 属性必须定义父元素上。通常 `perspective-origin` 属性本身不做任何事情，它必须被定义在设置 `perspective` 属性的元素上。换句话说，`perspective-origin` 属性需要与 `perspective` 属性结合起来使用，以便将视点移至元素的中心以外位置，如图 11-15 所示。

看一样东西不可能一直都在中心位置看，想换个角度时就离不开 `perspective-origin` 属性，来自于 W3C 官网的图可以简单阐述这一观点，如图 11-16 所示。

2. 实战体验：对立方体的影响

本示例演示了修改 `perspective-origin` 的属性值对立方体的影响。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>perspective-origin 属性的基本使用 </title>
<style type="text/css" media="screen">
.wrapper {
  width: 30%;
  display: inline-block;
  padding-bottom: 1em;
}
.wrapper h1 {
  text-align: center;
  font-size: 1.5em;
}
.cube {
  font-size: 2em;
  width: 2em;
  height: 2em;
  margin: .5em auto;
  transform-style: preserve-3d;
  perspective: 250px;
}
```

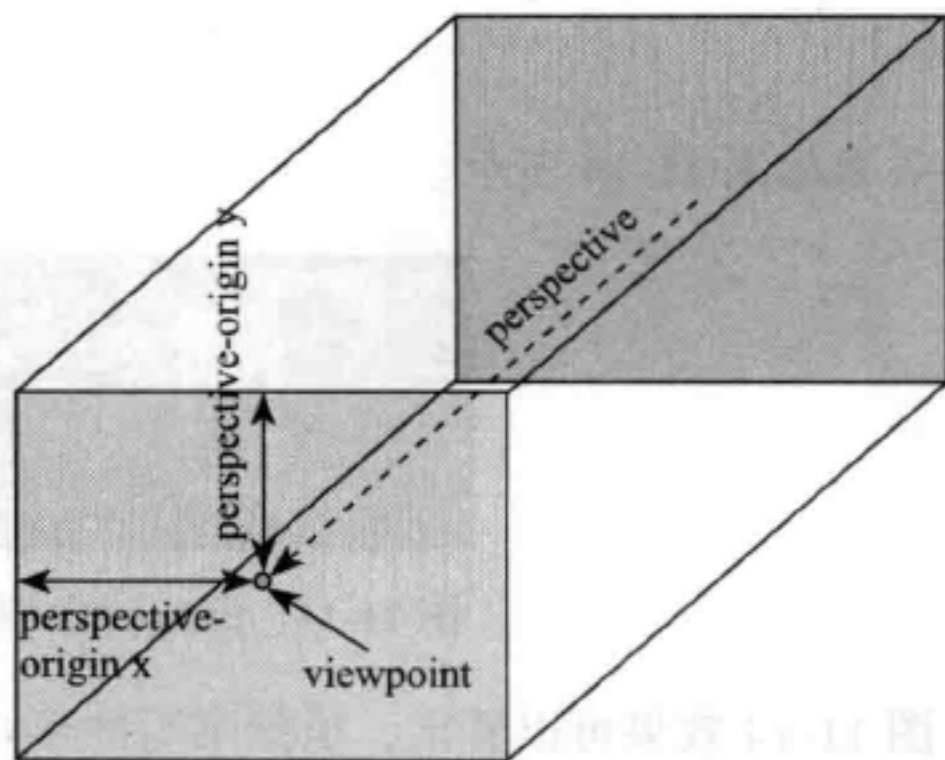


图 11-15 `perspective-origin` 与 `perspective` 结合使用示意图

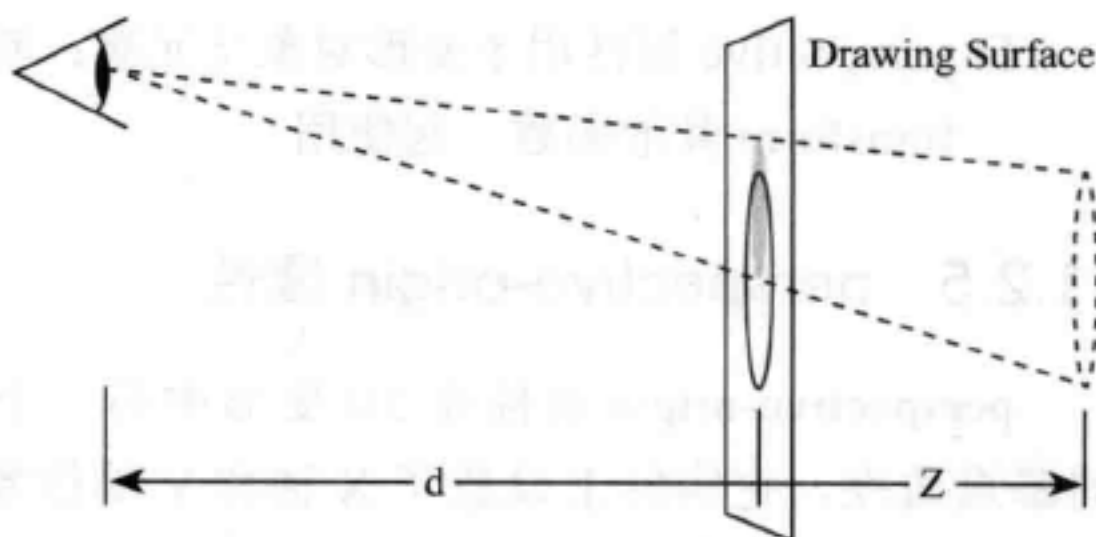


图 11-16 `perspective-origin` 向上移改变视角位置

```

}
.w1 .cube {
    perspective-origin: top left;
}
.w2 .cube {
    perspective-origin: top;
}
.w3 .cube {
    perspective-origin: top right;
}
.w4 .cube {
    perspective-origin: left;
}
.w5 .cube {
    perspective-origin: center;
}
.w6 .cube {
    perspective-origin: right;
}
.w7 .cube {
    perspective-origin: bottom left;
}
.w8 .cube {
    perspective-origin: bottom;
}
.w9 .cube {
    perspective-origin: bottom right;
}
.side {
    position: absolute;
    width: 2em;
    height: 2em;
    background: rgba(255, 99, 71, 0.6);
    border: 1px solid rgba(0, 0, 0, 0.5);
    color: white;
    text-align: center;
    line-height: 2em;
}
.front{
    transform:translateZ(1em);
}
.top{
    transform: rotateX( 90deg)  translateZ(1em);
}
.right{
    transform: rotateY( 90deg)  translateZ(1em);
}
.left{
    transform: rotateY(-90deg)  translateZ(1em);
}

```



```

.bottom {
  transform: rotateX(-90deg) translateZ(1em);
}
.back{
  transform: rotateY(-180deg) translateZ(1em);
}
</style>
</head>
<body>
<div class="wrapper w1">
  <h1><code>top left</code></h1>
  <div class="cube">
    <div class="side front">1</div>
    <div class="side back">6</div>
    <div class="side right">4</div>
    <div class="side left">3</div>
    <div class="side top">5</div>
    <div class="side bottom">2</div>
  </div>
</div>
<!-- w2~w9 结构与 w1 类似 -->
</body>
</html>

```

效果如图 11-17 所示。

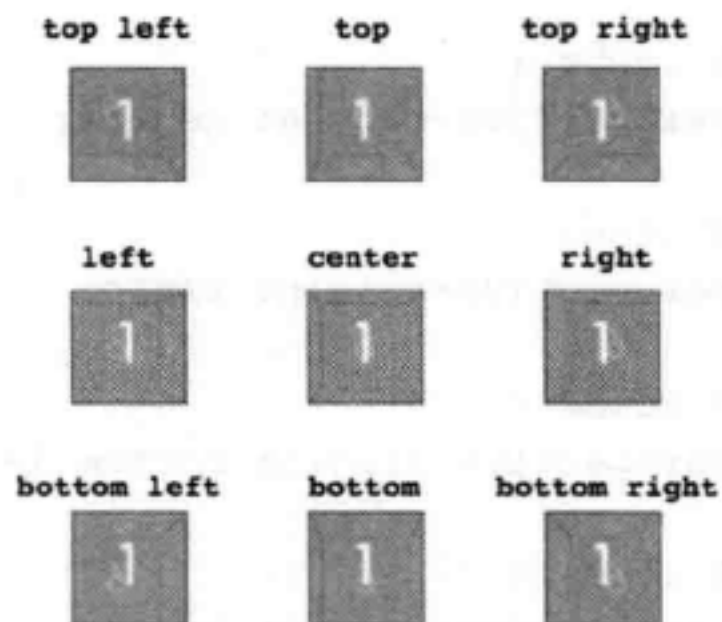


图 11-17 perspective-origin 取不同值对立方体影响

11.2.6 backface-visibility 属性

backface-visibility 属性决定元素旋转背面是否可见。对于未旋转的元素，该元素的正面面向观看者。当其 Y 轴旋转约 180 度时会导致元素的背面面对观众。

1. backface-visibility 属性的语法

backface-visibility 属性使用语法很简单。

```
backface-visibility: visible | hidden
```

该属性被设置为以下两个关键词之一。

❑ visible: 默认值，反面可见。

❑ hidden: 反面不可见。

一个元素的可见性与 backface-visibility:hidden 决定如下。

❑ 元素在 3D 环境下渲染上下文，将根据 3D 变形矩阵来计算，反之元素不在 3D 环境下渲染上下文，将根据 2D 变形矩阵来计算。

❑ 如果组件的矩阵在第 3 行、3 列是负值，元素反面是隐藏，反之是可见的。

简单来说，backface-visibility 属性可用于隐藏内容的背面。默认情况下，背面可见，这意味着即使在翻转后，旋转的内容仍然可见。当 backface-visibility 设置为 hidden 时，旋转后内容将隐藏，因为旋转后正面将不再可见。

2. 实战体验: hidden 和 visible 的区别

该功能可以模拟多面的对象, 例如下例中使用的纸牌。通过将 `backface-visibility` 设置为 `hidden`, 可以轻松确保只有正面可见。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>backface-visibility 属性的基本使用 </title>
<style type="text/css" media="screen">
  /* 基本样式省略 */
  .stage{
    float: left;
    border: 1px dotted orange;
    position: relative;
    margin: 20px;
    border-radius: 8px;
    perspective: 1000;
  }
  .container {
    width: 102px;
    height: 142px;
    position: relative;
    transition: 0.5s;
    transform-style: preserve-3d;
  }
  .card {
    position: absolute;
    top: 0;
    right: 0;
    bottom: 0;
    left: 0;
    backface-visibility: hidden;
  }

  .front {
    background: url(images/cardjfront.png) no-repeat center/100% 100%;
    z-index: 2;
  }
  .back {
    background: url(images/cardjback.png) no-repeat center/100% 100%;
    transform: rotateY(180deg);
  }
  .stage:nth-child(1) .container{
    transform: rotateY(0deg);
  }
  .stage:nth-child(2) .container{
    transform: rotateY(30deg);
  }

```

```

.stage:nth-child(3) .container{
  transform: rotateY(60deg);
}
.stage:nth-child(4) .container{
  transform: rotateY(90deg);
}
.stage:nth-child(5) .container{
  transform: rotateY(120deg);
}
.stage:nth-child(6) .container{
  transform: rotateY(150deg);
}
.stage:nth-child(7) .container{
  transform: rotateY(180deg);
}
</style>
</head>
<body>
  <div class="stage">
    <div class="container">
      <div class="card front"></div>
      <div class="card back"></div>
    </div>
  </div>
  <!-- 其他结构类似 -->
</body>
</html>

```

效果如图 11-18 所示。



图 11-18 backface-visibility 取值为 hidden 时各张扑克牌 Y 轴旋转效果

在本例中，未旋转时将看到扑克牌正面，也就是红桃 J，这是由于其定位于顶部。随着向扑克牌应用的旋转超过 90 度，第二个 div 的 backface-visibility:hidden 属性导致其不可见，因此将显示扑克牌正面。接下来可以将第二个 div 的 backface-visibility 设置为 visible。

```

.card {
  position: absolute;
  top: 0;
  right: 0;
  bottom: 0;
  left: 0;
  backface-visibility: visible;
}

```


效果如图 11-19 所示。



图 11-19 backface-visibility 取值为 visible 时各张扑克牌 Y 轴旋转效果

通过两个 3D 立方体来做一个实例，从视觉上更直观地区分 backface-visibility 取值为 hidden 和 visible 的区别。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>Backface Visibility</title>
<style type="text/css" media="screen">
.container {
  width: 500px;
  height: 300px;
  float: left;
  position: relative;
  margin: 30px;
  border: 1px solid #CCC;
  perspective: 1200px;
}
.cube {
  width: 100%;
  height: 100%;
  position: absolute;
  animation: spinCube 8s infinite ease-in-out;
  transform-style: preserve-3d;
  transform: translateZ( -100px );
}
@keyframes spinCube {
  0% { -moz-transform: translateZ( -100px )
        rotateX( 0deg ) rotateY( 0deg ); }
  100% { -moz-transform: translateZ( -100px )
        rotateX( 360deg ) rotateY( 360deg ); }
}

.side {
  display: block;
  position: absolute;
  width: 196px;
  height: 196px;
  border: 2px solid black;
  line-height: 196px;
```

```

font-size: 120px;
font-weight: bold;
color: white;
text-align: center;
}

.cube.backface-visibility-visible .side {
  backface-visibility: visible;
}
.cube.backface-visibility-hidden .side {
  backface-visibility: hidden;
}

.cube .front { background: hsla( 0, 100%, 50%, 0.7 ); }
.cube .back  { background: hsla( 60, 100%, 50%, 0.7 ); }
.cube .right { background: hsla( 120, 100%, 50%, 0.7 ); }
.cube .left  { background: hsla( 180, 100%, 50%, 0.7 ); }
.cube .top   { background: hsla( 240, 100%, 50%, 0.7 ); }
.cube .bottom { background: hsla( 300, 100%, 50%, 0.7 ); }

.cube .front {
  transform: translateZ( 100px );
}
.cube .back {
  transform: rotateX( -180deg ) translateZ( 100px );
}
.cube .right {
  transform: rotateY( 90deg ) translateZ( 100px );
}
.cube .left {
  transform: rotateY( -90deg ) translateZ( 100px );
}
.cube .top {
  transform: rotateX( 90deg ) translateZ( 100px );
}
.cube .bottom {
  transform: rotateX( -90deg ) translateZ( 100px );
}
</style>
</head>
<body>
  <div class="container">
    <h1>backface-visibility:visible</h1>
    <div class="cube backface-visibility-visible">
      <div class="side front">1</div>
      <div class="side back">2</div>
      <div class="side right">3</div>
      <div class="side left">4</div>
      <div class="side top">5</div>
      <div class="side bottom">6</div>
    </div>
  </div>

```

```

</div>

<div class="container">
  <h1>backface-visibility:hidden</h1>
  <div class="cube backface-visibility:hidden">
    <div class="side front">1</div>
    <div class="side back">2</div>
    <div class="side right">3</div>
    <div class="side left">4</div>
    <div class="side top">5</div>
    <div class="side bottom">6</div>
  </div>
</div>
</body>
</html>

```

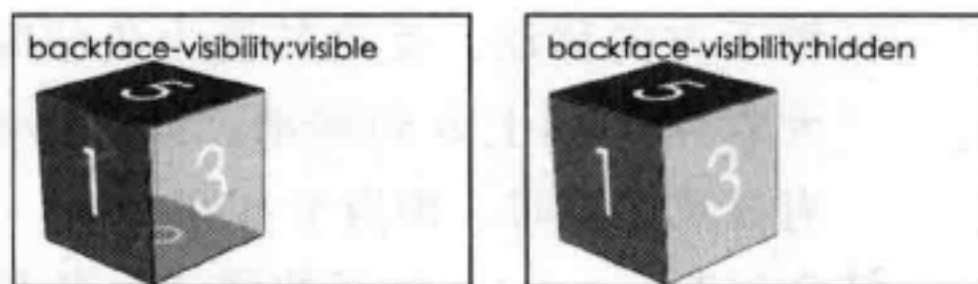


图 11-20 backface-visibility 取值为 visible 和 hidden 对 3D 立方体影响

效果如图 11-20 所示。左边立方体每个面都能看得到，而右边的立方体只能看到视力范围的面。

11.3 CSS3 2D 变形

在二维或三维空间，元素可以被扭曲、移位或旋转。只不过 2D 变形工作在 X 轴和 Y 轴，也就是大家常说的水平轴和垂直轴；而 3D 变形工作在 X 轴和 Y 轴之外，还有一个 Z 轴，这些 3D 变换不仅可以定义元素的长度和宽度，还有深度。首先讨论元素在 2D 平面如何变换，然后在进入 3D 变换的讨论。

CSS3 2D 变换让 Web 设计师有了更多的自由来装饰和变形 HTML 组件，同时有更多的功能装饰文本和更多的动画选项来装饰 div 元素。在 CSS3 2D 变形中主要包含的一些基本功能如下。

11.3.1 2D 位移

在这里 translate 是一种方法，将元素向指定的方向移动，类似于 position 中的 relative。可以简单理解为，使用 translate() 函数可以把元素从原来的位置移动，而不影响在 X、Y 轴上任何组件。

1. translate() 函数的语法

translate() 函数的使用语法如下。

```
translate(tx)
```

或者：

```
translate(tx,ty)
```


translate() 函数可以取一个值 tx，也可以取两个值 tx 和 ty，其取值具体说明如下（图 11-21）。

- tx：代表 X 轴（横坐标）移动的向量长度，当其值为正值时，元素向 X 轴右方向移动，反之其值为负值时，元素向 X 轴左方向移动。
- ty：代表 Y 轴（纵坐标）移动的向量长度，当其值为正值时，元素向 Y 轴下方向移动，反之其值为负值时，元素向 Y 轴上方向移动。如果 ty 没有显式设置时，相当于 ty=0。

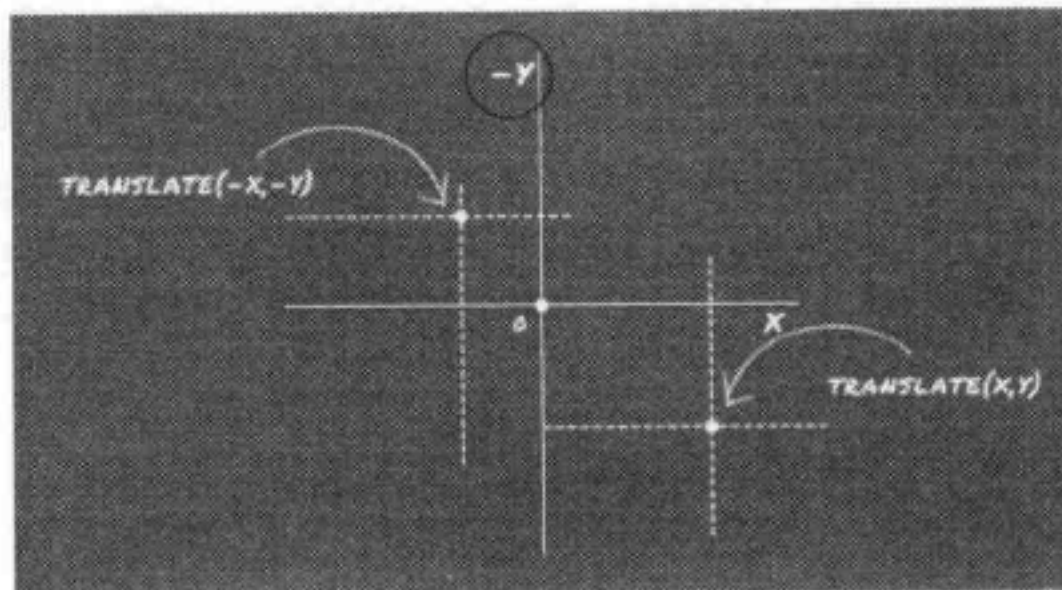


图 11-21 translate() 函数移动坐标示意图

结合起来，translate() 函数移动元素主要有以下三种移动。

- 水平移动：向右移动 translate(tx,0) 和向左移动 translate(-tx,0)。
- 垂直移动：向上移动 translate(0,-ty) 和向下移动 translate(0,ty)。
- 对角移动：右下角移动 translate(tx,ty)、右上角移动 translate(tx,-ty)、左上角移动 translate(-tx,-ty) 和左下角移动 translate(-tx,ty)。

2. 实战体验：移动纸牌

使用 transform:translate(tx,ty) 将一个对象从其原始位置移动，其中 tx 值为正值和 ty 值等于 0 时，对象向右移动。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>translate() 函数</title>
  <style type="text/css">
    div {
      width: 500px;
      height: 300px;
      margin: 30px auto;
      position: relative;
      background: url(images/bg-grid.jpg) no-repeat center center;
      background-size: 100% 100%;
    }
    div img {
      position: absolute;
      top: 50%;
      left: 50%;
      margin-left: -35px;
      margin-top: -50px;
    }
    div img:nth-child(1){
```

```

        opacity: .5;
        z-index: 1;
    }
    div img:nth-child(2){
        opacity: 1;
        z-index: 2;
        transform: translate(100px,0);
    }

</style>
</head>
<body>
    <div>
        
        
    </div>
</body>
</html>

```

效果如图 11-22 所示。

在这个示例中，让扑克牌梅花 K 相对于原点中心位置向右移动 100px。如果仅需让元素向右移动，可以省略 ty 值。换句话说 ty 值为 0 时可以省略不写。

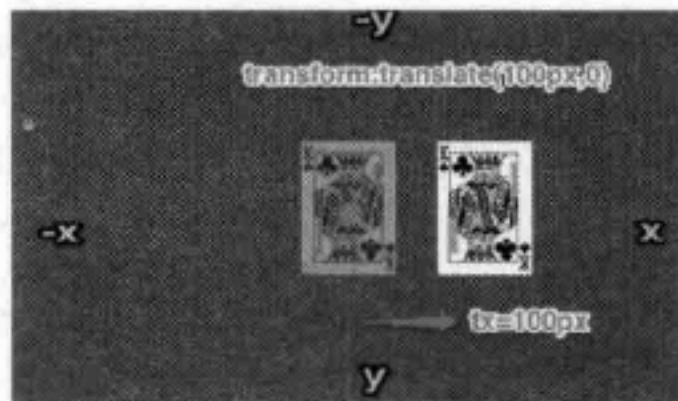


图 11-22 translate() 函数让元素向右移动

```

div img:nth-child(2){
    opacity: 1;
    z-index: 2;
    transform: translate(100px);
}

```

要将一个对象移动到左边，只需要输入一个负数的 X 轴坐标，而 Y 坐标应保持 0，基于前面的实例，将扑克牌向左边移动 100px。

```

div img:nth-child(2){
    transform: translate(-100px,0);
}

```

效果如图 11-23 所示。

垂直移动一个对象很简单，几乎和水平移动对象相同。唯一的区别是，将使用 Y 轴的值控制对象向上和向下移动位移量。正如前面提到的，Y 轴的坐标值为正值时，对象向下移动；反之其坐标值为负值时，对象向上移动。而 X 轴的坐标值应该保持为 0。来看一个简单的实例，将一扑克牌向上，向下移动 100px。

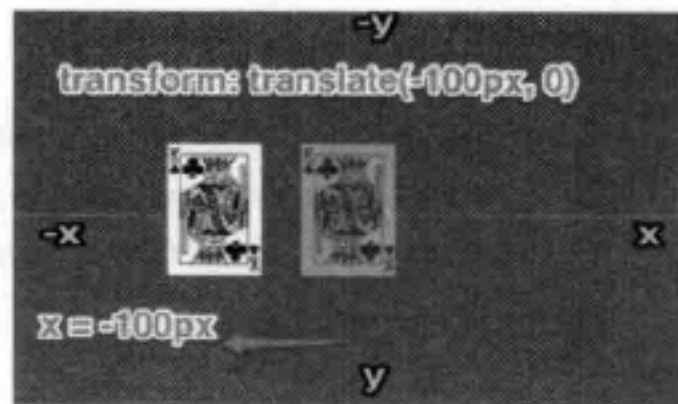


图 11-23 translate() 函数让元素向左移动

```

<!DOCTYPE HTML>
<html lang="en-US">
<head>

```

```

<meta charset="UTF-8">
<title>translate() 函数</title>
<style type="text/css">
  div {
    width: 500px;
    height: 500px;
    margin: 30px auto;
    position: relative;
    background: url(images/bg-grid.jpg) no-repeat center center;
    background-size: 100% 100%;
  }
  div img {
    position: absolute;
    top: 50%;
    left: 50%;
    margin-left: -35px;
    margin-top: -50px;
  }
  div img:nth-child(1){
    opacity: .5;
    z-index: 1;
  }
  div img:nth-child(2){
    z-index: 2;
    transform: translate(0,-100px);
  }
  div img:nth-child(3){
    z-index: 3;
    transform: translate(0,100px);
  }
</style>
</head>
<body>
  <div>
    
    
    
  </div>
</body>
</html>

```

效果如图 11-24 所示。

要让一个元素对角移动，需要结合 X 轴和 Y 轴两坐标的值。根据不同的方向，X 轴和 Y 轴的值可能是正值或是负值。如果要将一张扑克牌向右上角移动，需要将 X 轴坐标设置为正值，将 Y 轴坐标设置为负值；如果

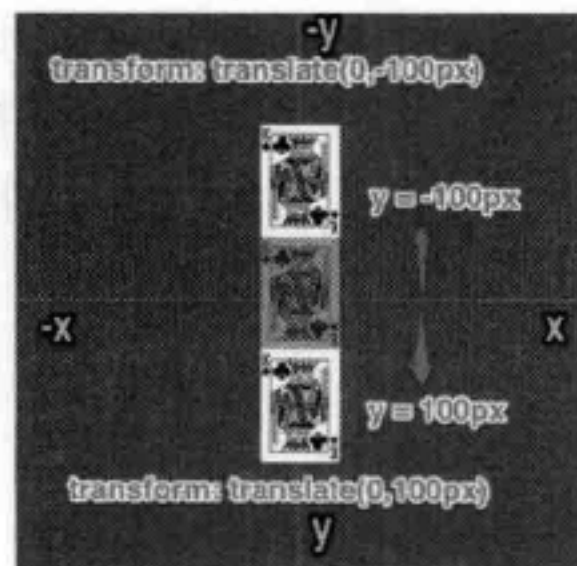
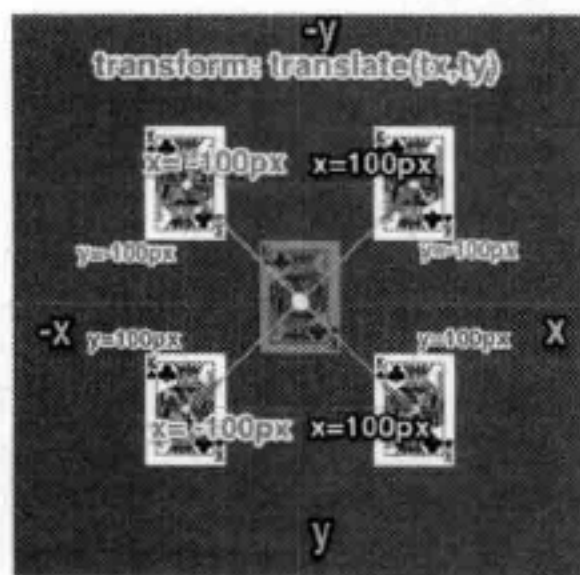


图 11-24 translate() 函数让元素向上、向下移动

要将一张扑克牌向右下角移动, 需要将 X、Y 轴坐标设置为正值; 如果要将一张扑克牌向左上角移动, 需要将 X、Y 轴坐标设置为负值; 如果要将扑克牌向左下角移动, 需要将 X 坐标设置为负值, Y 轴坐标设置为正值。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>translate() 函数</title>
  <style type="text/css">
    div {
      width: 500px;
      height: 500px;
      margin: 30px auto;
      position: relative;
      background: url(images/bg-grid.jpg) no-repeat center center;
      background-size: 100% 100%;
    }
    div img {
      position: absolute;
      top: 50%;
      left: 50%;
      margin-left: -35px;
      margin-top: -50px;
    }
    div img:nth-child(1){
      opacity: .5;
      z-index: 1;
    }
    div img:nth-child(2){
      z-index: 2;
      transform: translate(100px,-100px);
    }
    div img:nth-child(3){
      z-index: 3;
      transform: translate(100px,100px);
    }
    div img:nth-child(4){
      z-index: 3;
      transform: translate(-100px,-100px);
    }
    div img:nth-child(5){
      z-index: 3;
      transform: translate(-100px,100px);
    }
  </style>
</html>
```

...



效果如图 11-25 所示。

图 11-25 translate() 函数向对角移动元素

如果要将对象沿着一个方向移动，如沿着水平轴或者纵轴移动，可以使用 `translate(tx,0)` 和 `translate(0,ty)` 来实现。其实在变形中还为单独一个方向移动对象提供了更简单的方法。

□ `translateX()`：水平方向移动一个对象。通过给定一个 X 轴方向的数值指定对象沿水平轴方向的位移。简单点说，对象只向 X 轴进行移动，如果值为正值，对象向右移动；如果值为负值，对象向左移动。

□ `translateY()`：纵轴方向移动一个对象。通过给定一个 Y 轴方向的数值指定对象沿纵轴方向的位移。简单点说，对象只向 Y 轴进行移动，如果值为正值，对象向下移动；如果值为负值，对象向上移动。

这两个函数和前面介绍的 `translate()` 函数不同的是每个方法只接受一个值。

□ `transform:translate(-100px,0)` 实际上等于 `transform:translateX(-100px)`。

□ `transform:translate(0,-100px)` 实际上等于 `transform:translateY(-100px)`。

11.3.2 2D 缩放

缩放函数 `scale()` 让元素根据中心原点对对象进行缩放，默认值为 1。因此 0.01 到 0.99 之间的任何值，使一个元素缩小；而任何大于或等于 1.01 的值，让元素显得更大。如图 11-26 所示。

1. `scale()` 函数的语法

缩放 `scale()` 函数和 `translate()` 函数的语法非常相似，可以接受一个值，也可以接受两个值，只有一个值时，其第二个值默认与第一个值相等。例如，`scale(1,1)` 元素不会有任何变化，而 `scale(2,2)` 让元素沿 X 轴和 Y 轴放大两倍。其使用语法如下。

```
scale(sx)
```

或者：

```
scale(sx,sy)
```

其取值简单说明如下。

□ `sx`：指定横向坐标（X 轴）方向的缩放向量，如果值为 0.01 ~ 0.99 之间，会让对象在 X 轴方向缩小，如果值大于或等于 1.01，对象在 X 轴方向放大。

□ `sy`：指定纵向坐标（Y 轴）方向的缩放量，如果值为 0.01 ~ 0.99 之间，会让对象在 Y 轴方向缩小，如果值大于或等于 1.01，对象在 Y 轴方向放大。

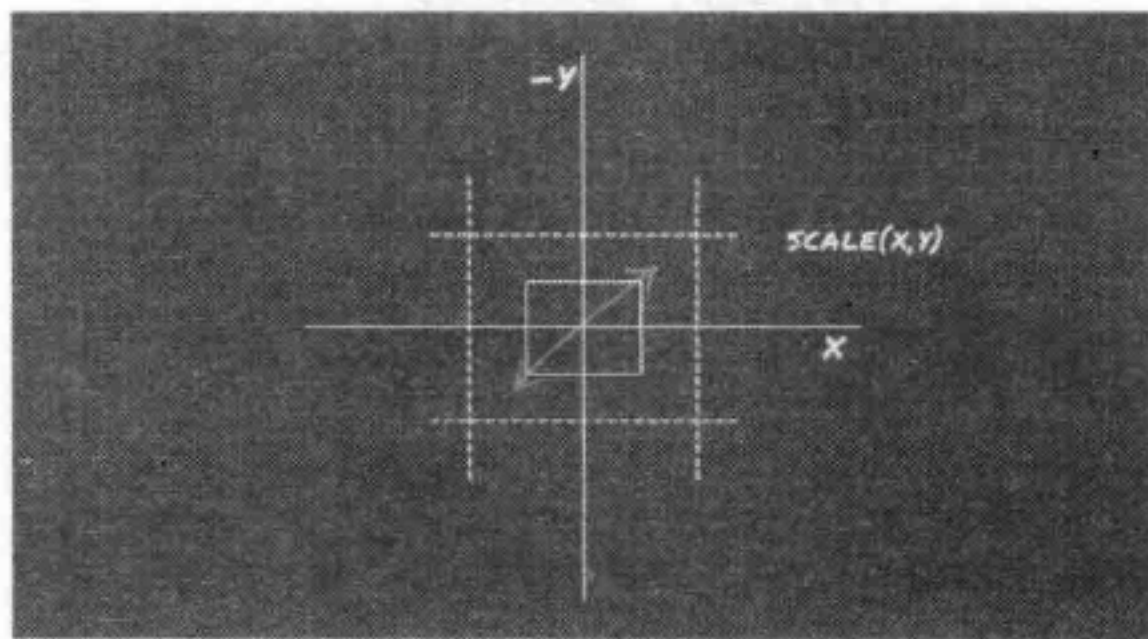


图 11-26 `scale()` 函数缩放示意图

2. 实战体验：缩放纸牌

这有一个简单的实例如下。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>scale() 函数</title>
  <style type="text/css">
    div {
      width: 500px;
      height: 500px;
      margin: 30px auto;
      position: relative;
      background: url(images/bg-grid.jpg) no-repeat center center;
      background-size: 100% 100%;
    }
    div img {
      position: absolute;
      top: 50%;
      left: 50%;
      margin-left: -35px;
      margin-top: -50px;
    }
    div img:nth-child(1){
      opacity: .8;
      z-index: 2;
      border: 1px solid red;
    }
    div img:nth-child(2){
      z-index: 1;
      transform: scale(1.5);
    }
  </style>
</head>
<body>
  <div>
    
    
  </div>
</body>
</html>
```

效果如图 11-27 所示。

右面的例子将扑克牌放大了 1.5 倍或是实际尺寸的 150%。因为同时对 X 和 Y 轴方向放大，所以只需要给 scale() 声明一个值。也可以使用 transform:scale(1.5,1.5) 实现相同的效果。



图 11-27 scale() 函数放大元素

此外如果要缩小一个元素，会专门使用一个 0 ~ 0.9999 的值，像下面的例子，将扑克牌缩放一半，也就是实际尺寸的 50%。

```
div img:nth-child(2){
  z-index: 2;
  transform: scale(.5);
}
```

效果如图 11-28 所示。

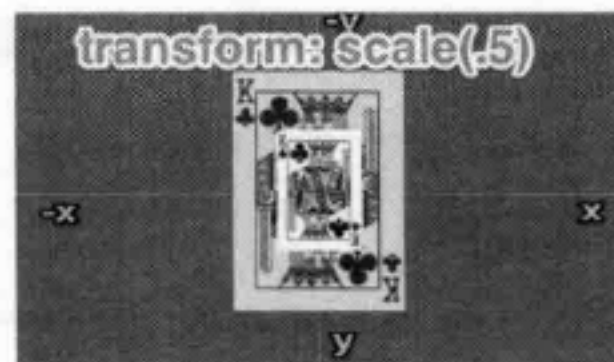


图 11-28 scale() 函数缩小元素

但是，如果将值设置为 0 时，元素将会消失。如果没必要，不会这样做的。仅给 scale() 函数只显式设置一个值时，会使对象成正比例放大或缩小。如果需要将对象在 X 轴和 Y 轴两个方向设置不同的值。

```
div img:nth-child(2){
  z-index: 2;
  transform: scale(.5,1.2);
}
```

效果如图 11-29 所示。

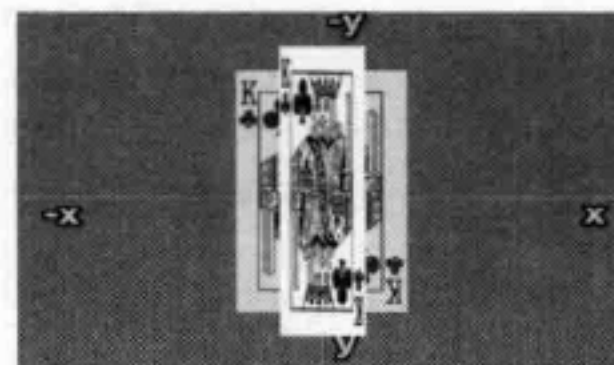


图 11-29 scale() 缩放元素

scale() 函数和 translate() 函数极其相似，除了能通过 scale() 函数使用元素水平方向和垂直方向同时缩放（也就是元素沿 X 轴和 Y 轴同时缩放）之外，也可以使元素仅沿着 X 轴或 Y 轴方向缩放。

□ scaleX(): 相当于 scale(sx,1)。表示元素只在 X 轴（水平方向）缩放元素，默认值是 1。

□ scaleY(): 相当于 scale(1,sy)。表示元素只在 Y 轴（纵横方向）缩放元素，默认值是 1。

通过上面的介绍，不由想起图形编辑软件中的缩放工具，实现缩放对象的效果，如图 11-30 所示。在 CSS3 中的 scale() 函数和图形编辑软件中的缩放工具几乎相同。



图 11-30 CSS3 的 scale() 函数对比图形编辑软件的缩放工具

在 `scale()` 函数中, 取值除了可以取正值, 还可以取负值。只不过取负值时, 会先让元素进行翻转, 然后再进行缩放。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>scale() 函数</title>
  <style type="text/css">
    .wrapper {
      width: 500px;
      height: 400px;
      margin: 30px auto;
      position: relative;
      background: url(images/bg-grid.jpg) no-repeat center center;
      background-size: 100% 100%;
    }
    .wrapper > div {
      position: absolute;
      background-color: hsla(220,20%,20%,.3);
      top: 50%;
      left: 50%;
      margin-left: -100px;
      margin-top: -100px;
      width: 198px;
      height: 198px;
      border: 1px dotted orange;
      text-align: center;
      line-height: 198px;
      color: #fff;
      font-size: 20px;
      transform: scale(-1.5);
    }
  </style>
</head>
<body>
  <div class="wrapper">
    <div>Scale(-1.5)</div>
  </div>
</body>
</html>
```



图 11-31 `scale()` 函数取值为负数时效果

效果如图 11-31 所示。

`scale()` 函数对元素进行缩放时, 都是以元素的中心为基点, 但可以通过 `transform-origin` 来改变元素的基点。

11.3.3 2D 旋转

旋转函数 `rotate()` 通过指定的角度参数对元素根据对象原点指定一个 2D 旋转。主要在二维空间内进行操作，接受一个角度值，用来指定旋转的幅度。如果这个值为正值，元素相对原点中心顺时针旋转；如果这个值为负值，元素相对原点中心逆时针旋转。如图 11-32 所示。

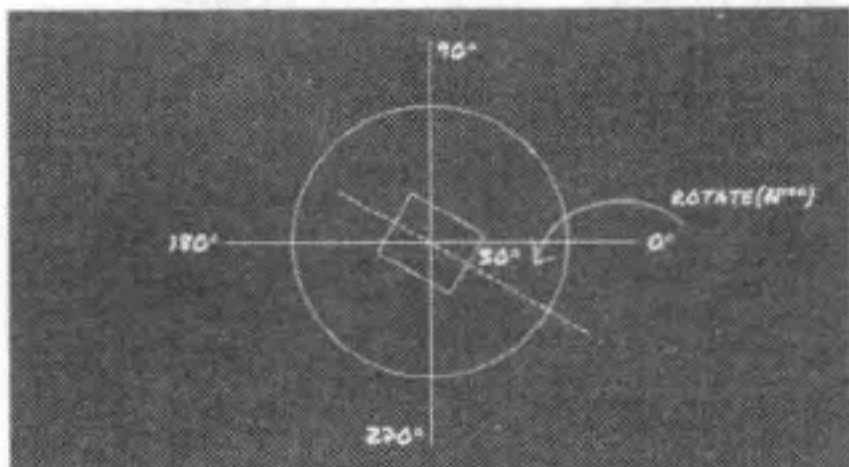


图 11-32 `rotate()` 函数旋转元素示意图

1. `rotate()` 函数的语法

`rotate()` 函数的使用很简单，其基本语法如下。

```
rotate(a);
```

`rotate()` 函数只接受一个值 `a`，代表旋转的角度值。其取值可以是正的，也可以是负的。

□ 取值为正值时，元素默认相对元素中心点顺时针旋转。

□ 取值为负值时，元素默认相对元素中心点逆时针旋转。

2. 实战体验：旋转纸牌

看一个简单的例子，扑克牌相对于元素中心点顺时针旋转 45 度。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>rotate() 函数</title>
  <style type="text/css">
    div {
      width: 500px;
      height: 300px;
      margin: 30px auto;
      position: relative;
      background: url(images/bg-grid.jpg) no-repeat center center;
      background-size: 100% 100%;
    }
    div img {
      position: absolute;
      top: 50%;
      left: 50%;
      margin-left: -70px;
      margin-top: -100px;
    }
    div img:nth-child(1){
      z-index: 1;
      opacity: .6;
    }
    div img:nth-child(2){
```



```

        z-index: 2;
        transform: rotate(45deg);
    }
</style>
</head>
<body>
    <div>
        
        
    </div>

</body>
</html>

```

效果如图 11-33 所示。

在默认情况下，rotate() 函数旋转元素是相对于元素中心点进行旋转，同样可以通过 transform-origin 属性重置元素的旋转原点。

```

div img:nth-child(2){
    z-index: 2;
    transform-origin: top left;
    transform: rotate(45deg);
}

```

基于上例，修改旋转原点后的效果就完全不同了，如图 11-34 所示。

rotate() 函数也同样可以和图形编辑软件中的旋转工具的功能对比起来理解。是 CSS3 中 rotate() 函数在 2D 中的旋转与 Photoshop 制作软件中旋转工具的对比。如图 11-35 所示。



图 11-33 rotate(45deg) 使用扑克牌顺时针旋转 45 度

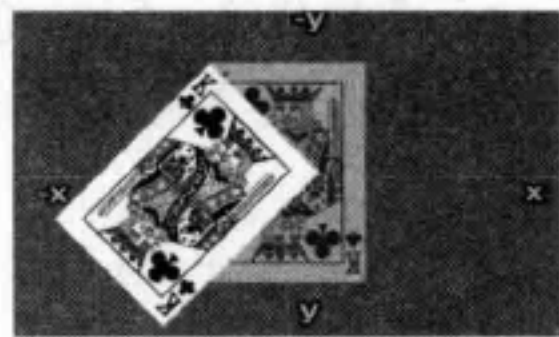


图 11-34 修改旋转原点后 rotate(45deg) 的效果

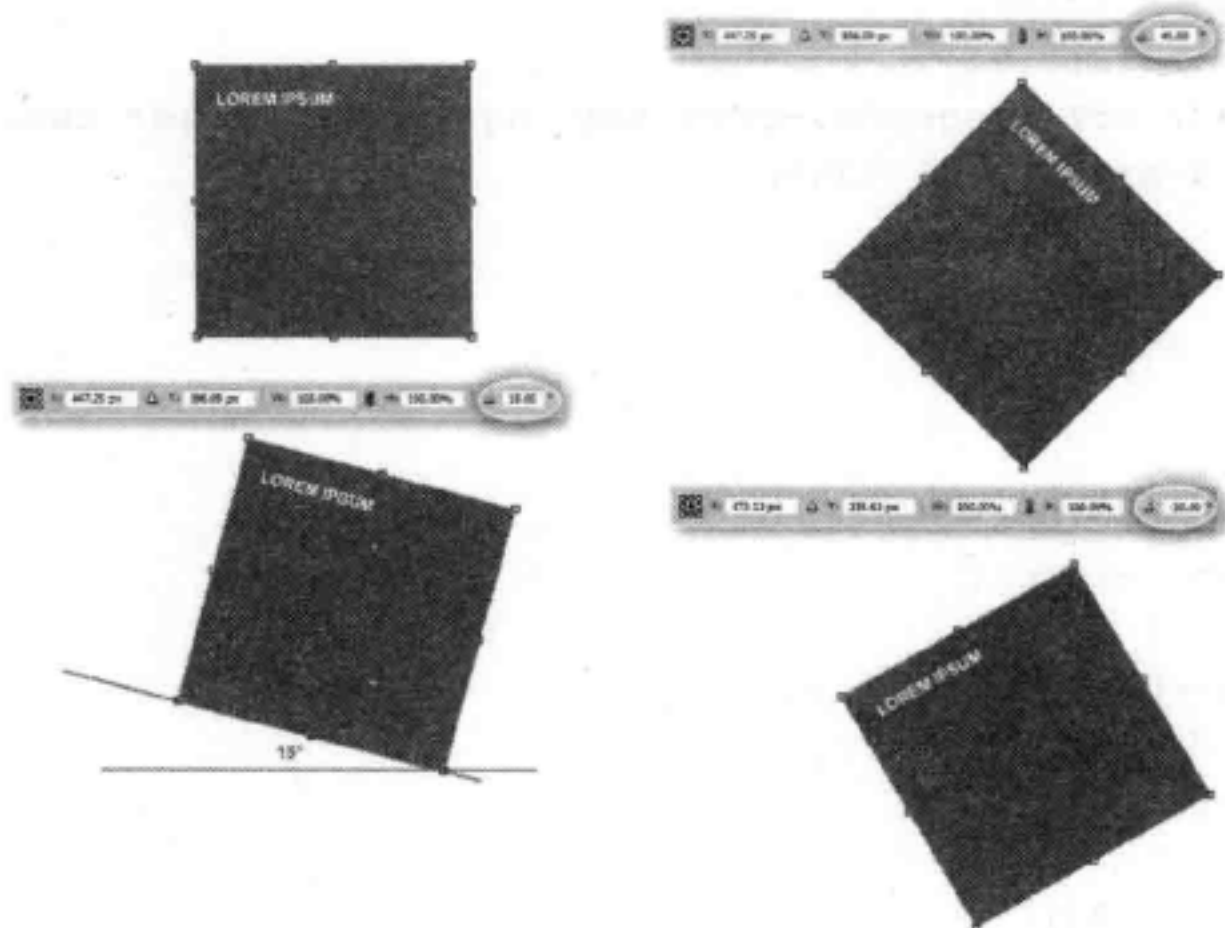


图 11-35 CSS3 中 2D 旋转 rotate() 函数与 Photoshop 制作编辑软件中的旋转工具

11.3.4 2D 倾斜

倾斜函数 `skew()` 能够让元素倾斜显示，可以将一个对象以其中心位置围绕着 X 轴和 Y 轴按照一定的角度倾斜。与 `rotate()` 函数的旋转不同，`rotate()` 函数只是旋转，而不会改变元素的形状。`skew()` 函数不会旋转，而只会改变元素的形状。

1. `skew()` 函数的语法

`skew()` 函数语法格式如下。

```
skew(ax)
```

或者：

```
skew(ax, ay)
```

其属性值说明如下，如果未显式设置这个值，其默认为 0。

❑ `ax`：指定元素水平方向（X 轴方向）倾斜的角度。

❑ `ay`：指定元素垂直方向（Y 轴方向）倾斜的角度。

2. 实战体验：倾斜纸牌

这里有一个简单的例子。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>skew() 函数</title>
  <style type="text/css">
    div {
      width: 500px;
      height: 300px;
      margin: 30px auto;
      position: relative;
      background: url(images/bg-grid.jpg) no-repeat center center;
      background-size: 100% 100%;
    }
    div img {
      position: absolute;
      top: 50%;
      left: 50%;
      margin-left: -70px;
      margin-top: -100px;
    }
    div img:nth-child(1){
      z-index: 1;
      opacity: .6;
    }
    div img:nth-child(2){
```

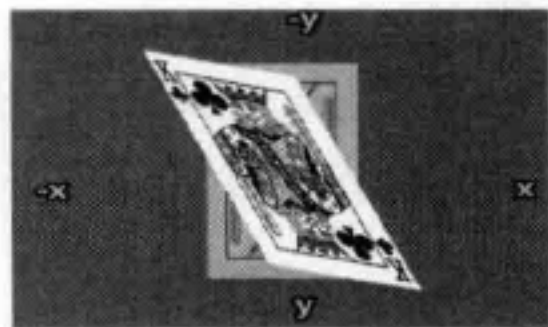
```

    z-index: 2;
    transform: skew(30deg,10deg);
}

</style>
</head>
<body>
  <div>
    
    
  </div>

</body>
</html>

```



效果如图 11-36 所示。

skew() 函数和 CSS3 变形中的 translate()、scale() 函数 图 11-36 skew() 函数让元素倾斜一样，除了可以使用 skew(tx,ty) 函数让元素相于元素中心为原点在 X 轴和 Y 轴倾斜之外，还可以使用 skewX() 和 skewY() 函数让元素只在水平或垂直方向倾斜。

❑ skewX()：相当于 skew(ax,0) 和 skew(ax)。按给定的角度沿 X 轴指定一个倾斜变形。

skewX() 使元素以其中心为基点，并在水平方向（X 轴）进行倾斜变形。

❑ skewY()：相当于 skew(0,ay)。按给定的角度沿 Y 轴指定一个倾斜变形。skewY() 用来设置元素以其中心为基点并按给定的角度在垂直方向（Y 轴）倾斜变形。

在默认情况下，skew() 函数都是以元素的原中心点对元素进行倾斜变形，但是同样可以根据 transform-origin 属性，重新设置元素基点对元素进行倾斜变形。另外，skew() 函数和制图软件中的变形工具所起作用类似，如图 11-37 所示。

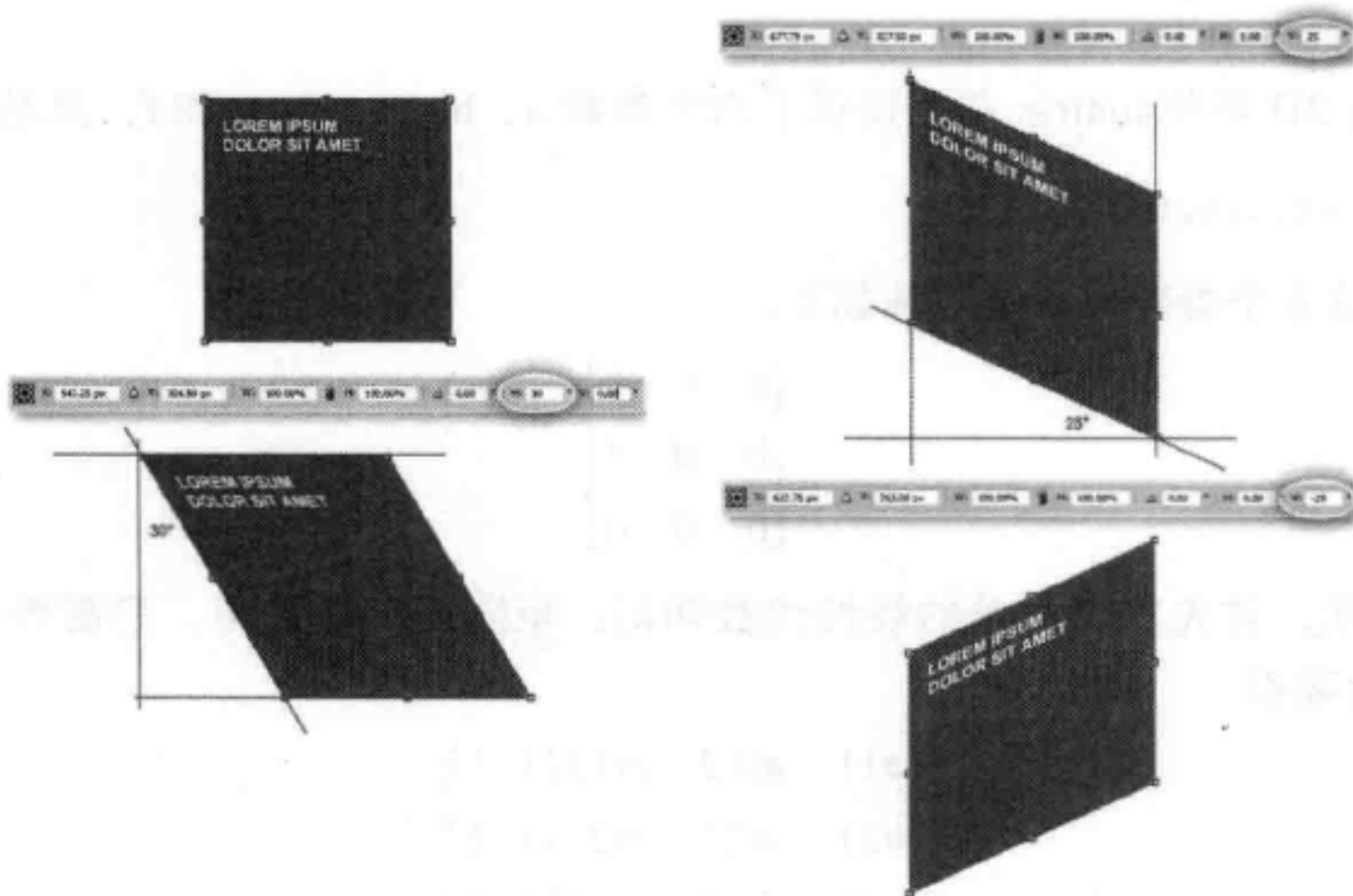


图 11-37 CSS3 的 skew() 函数与 Photoshop 的变形工具对比

11.3.5 2D 矩阵

CSS3 中 Transform 让操作变形变得很简单，如位移函数 `translate()`、缩放函数 `scale()`、旋转函数 `rotate()` 和倾斜函数 `skew()`。这几个函数很简单，也很方便，但是变形中的矩阵函数 `matrix()` 不常用。

当然，Web 设计师可以使用 `rotate()`、`skew()`、`scale()` 和 `translate()` 函数来满足它们的变形需要，那为什么要使用矩阵 `matrix()` 呢？在 CSS3 中的变形函数都可以使用 `matrix()` 函数来代替。

1. `matrix()` 函数的语法

使用矩阵可以实现一个复杂的 2D 变形，语法如下。

```
#object {
  transform-origin: 0 0;
  transform: rotate(15deg) translateX(230px) scale(1.5, 2.6) skew(220deg, -150deg)
  translateX(230px);
}
```

使用一个矩阵 `matrix()` 规则变成如下。

```
#object {
  transform-origin: 0 0;
  transform: matrix(1.06, 1.84, 0.54, 2.8, 466px, 482px);
}
```

也许很讨厌 `matrix()` 函数中的一大堆数字，但是要明白 CSS3 变形中的 `matrix()` 函数，先要了解矩阵 `matrix` 是怎么回事。接下来探讨 transform 中的 `matrix`。

CSS3 中变形的 `matrix` 分为两种，一种是 2D 矩阵，另外一种 3D 矩阵。先从简单的手——2D 矩阵 `matrix`。

CSS3 中的 2D 矩阵 `matrix` 总共提供了六个参数 `a`、`b`、`c`、`d`、`e` 和 `f`，其基本写法如下。

```
matrix(a,b,c,d,e,f)
```

实际上，这 6 个参数对应的矩阵如下。

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

在开始之前，首先复习简单的线性代数知识：矩阵与向量乘法。只需要了解三维向量与 3×3 矩阵的乘积。

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

$$=[v1*m11+v2*m12+v3*m13 \quad v1*m21+v2*m22+v3*m23 \quad v1*m31+v2*m32+v3*m33]$$

弄明白 3×3 的矩阵之后, 可知道 matrix 计算方法。x 和 y 是元素初始原点的坐标, x' 和 y' 则是通过矩阵变换后得到的新原点坐标。通过中间 3×3 变换矩阵, 对原先的坐标施加变换, 就能得到新的坐标了。依据矩阵变换规则即可得到:

$$x' = ax + cy + e$$

$$y' = bx + dy + f$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + cy + e \\ bx + dy + f \\ 1 \end{bmatrix}$$

为了更好地理解, 看一个简单的例子。

2. 实战体验: 变换矩阵

假设矩阵参数如下。

```
transform: matrix(1,0,0,1,50,50); /*a=1,b=0,c=0,d=1,e=50,f=50*/
```

现在, 根据这个矩阵偏移元素的中心点, 假设是 (0,0), 即 $x=0$, $y=0$ 。

可以按照上面介绍的矩阵方式, 将这个列成矩阵, 于是, 变换后的原点位置 x' 和 y' 可以根据矩阵向量的计算规则计算出来。如图 11-38 所示。

$$\begin{array}{ccc} \begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} & & \begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + cy + e \\ bx + dy + f \\ 0 + 0 + 1 \end{bmatrix} \\ \downarrow & & \\ \begin{bmatrix} 1 & 0 & 50 \\ 0 & 1 & 50 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} & & \begin{bmatrix} 1 & 0 & 50 \\ 0 & 1 & 50 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \times 0 + 0 \times 0 + 50 \\ 0 \times 0 + 1 \times 0 + 50 \\ 0 + 0 + 1 \end{bmatrix} = \begin{bmatrix} 50 \\ 50 \\ 1 \end{bmatrix} \end{array}$$

图 11-38 matrix(1,0,0,1,50,50) 的矩阵排列

计算得出 $x'=50$, $y'=50$, 即元素的原点由 (0,0) 移动到 (50,50) 的位置。实际上 transform:matrix(1,0,0,1,50,50) 就等同于 transform: translate(50px,50px)。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>matrix() VS translate()</title>
  <style type="text/css">
    div {
      width: 300px;
      height: 300px;
      margin: 30px auto;
      position: relative;
      background: url(images/bg-grid.jpg) no-repeat center center;
      background-size: 100% 100%;
```

```

    }
    div img {
      position: absolute;
      top: 50%;
      left: 50%;
      margin-left: -35px;
      margin-top: -50px;
    }
    div img:nth-child(1){
      z-index: 1;
      opacity: .6;
    }
    .matrix img:nth-child(2){
      z-index: 2;
      transform: matrix(1,0,0,1,50,50);
    }
    .translate img:nth-child(2){
      z-index: 2;
      transform: translate(50px,50px);
    }
  }

</style>
</head>
<body>
  <div class="matrix">
    
    
  </div>

  <div class="translate">
    
    
  </div>

</body>
</html>

```

效果如图 11-39 所示。

3. matrix() 和各变形函数之间的关系

前面通过一个简单的实例用 matrix() 函数实现 translate() 的位移效果。接下来分别看看 CSS3 变形中 matrix() 和各变形函数之间的关系。

(1) 位移的 matrix() 矩阵

首先看最简单的位移 translate() 函数。transform:translate(tx,ty) 的基本含义是将一个元素的显示位置平移 tx,ty。在矩阵变形中, translate 的 matrix 参数如下。

matrix(1,0,0,1,tx,ty)/*tx,ty 分别对应 X 和 Y 轴的增量*/



图 11-39 matrix(1,0,0,1,50,50) 和 translate(50px,50px) 效果对比

其对应的矩阵如下。

$$\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = [x + tx \quad y + ty \quad 1]$$

由此公式可知: `transform:translate(100px,100px)` 对应 `transform:matrix(1,0,0,1,100,100)`。
推算出的结果:

$$x' = x + tx = x + 100$$

$$y' = y + ty = y + 100$$

(2) 缩放的 `matrix()` 矩阵

`transform:scale(sx,sy)` 将一个元素按指定的倍数进行缩放如下。

`matrix(sx*x,0,0,sy*y,0,0);` /*sx 和 sy 分别对应 X 轴和 Y 轴的缩放比率 */

其对应的矩阵如下。

$$\begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = [x \times sx \quad y \times sy \quad 1]$$

由此公式可知: `transform:scale(1.5,1.5)` 对应 `transform:matrix(1.5,0,0,1.5,0,0)`。
推算出的结果:

$$x' = x \times sx = 1.5 \times x$$

$$y' = y \times sy = 1.5 \times y$$

(3) 旋转的 `matrix()` 矩阵

`transform:rotate(a)` 将一个元素按指定的角度旋转是。

`matrix(cos(a),sin(a),-sin(a),cos(a),0,0);` /*cos(a) 和 sin(a) 是指旋转度转 */

其对应的矩阵如下。

$$\begin{bmatrix} \cos a & -\sin a & 0 \\ \sin a & \cos a & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = [\cos a \times x - \sin a \times y \quad \sin a \times x + \cos a \times y \quad 1]$$

由此公式可知: `transform:rotate(45deg);` 对应 `transform:matrix(0.53,0.85,-0.85,0.53,0,0);`
[sin(45')=0.85,cos(45')=0.53]

推算出来的结果:

$$x' = \cos(a) x - \sin(a) y = \cos(45) x - \sin(45) y$$

$$y' = \sin(a) x + \cos(a) y = \sin(45) x + \cos(45) y$$

(4) 倾斜的 `matrix()` 矩阵

`transform:skew(ax,ay)` 将一个元素按指定的角度在 X 轴和 Y 轴倾斜。

`matrix(1,tan(ay),tan(ax),1,0,0);` /*tan(ax) 和 tan(ay) 是对应的倾斜角度 */

其对应的矩阵如下。

$$\begin{bmatrix} 1 & \tan(ax) & 0 \\ \tan(ay) & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = [x + \tan(ax) \times y \quad \tan(ay) \times x + y \quad 1]$$

由此公式可知: transform:skew(45deg) 对应 transform:matrix(1,0,1,1,0,0)。其中 $\tan(45^\circ)=1$ 。

推算出来的结果:

$$x' = x + \tan(a) \times y$$

$$y' = \tan(a) \times x + y$$

上面演示的是 CSS3 中常见的变形与矩阵的关系,除了上面演示的之外,还有其他的一些。所有 CSS3 变换与矩阵等价的关系如图 11-40 所示。

scale(a)	scaleX(x)	scaleY(y)	translateX(x)	translateY(y)	translate(x, y)
$\begin{pmatrix} a & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} x & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & y & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & x \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & y \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{pmatrix}$

skewX(x)	skewY(y)	skew(x, y)	rotate(θ)
$\begin{pmatrix} 1 & \tan x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ \tan y & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & \tan x & 0 \\ \tan y & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$

图 11-40 CSS3 所有变形与矩阵的等价关系

在制图软件中变形工具除了旋转、倾斜、位移、缩放等还有镜向(水平镜向、垂直镜向),如图 11-41 所示。



图 11-41 Photoshop 中的镜向翻转

但镜像对称在 CSS3 变形中没有相应的简化操作。只能通过矩阵 `matrix()` 来实现。通过前面的内容介绍,清楚地知道,元素变形的原点是其中心点(在没有显式重置之外),那么这个镜向的原点也不例外。因为该轴永远经过原点,因此,任意对称轴都可以用 $y=k*x$ 直线表示。则 `matrix` 表示如下。

```
matrix((1-k^2)/(1+k^2), 2k/(1+k^2), 2k/(1+k^2), (k^2-1)/(1+k^2), 0, 0)
```

这如何得到的呢?看看实现的过程。如图 11-42 所示, $y=k*x$, 并且知道点 (x,y) , 求其对称点 (x', y') 的坐标位置。

很简单,一是垂直,二是中心点在轴线上,因此有:

$$\begin{aligned} (y-y') / (x - x') &= -1/k & \rightarrow & ky - ky' = -x + x' \\ (x + x') / 2 * k &= y + y' & \rightarrow & kx + kx' = y + y' \end{aligned}$$

很简单的,把 x' 和 y' 提出来,就有:

$$\begin{aligned} x' &= (1-k^2)/(k^2+1) * x + 2k/(k^2+1) * y; \\ y' &= 2k/(k^2+1) * x + (k^2-1)/(k^2+1) * y; \end{aligned}$$

再结合矩阵公式:

$$\begin{aligned} x' &= ax+cy+e; \\ y' &= bx+dy+f; \end{aligned}$$

就可以得到:

$$\begin{aligned} a &= (1-k^2)/(k^2+1); \\ b &= 2k/(k^2+1); \\ c &= 2k/(k^2+1); \\ d &= (k^2-1)/(k^2+1); \end{aligned}$$

也就是上面 `matrix` 矩阵中的参数值。

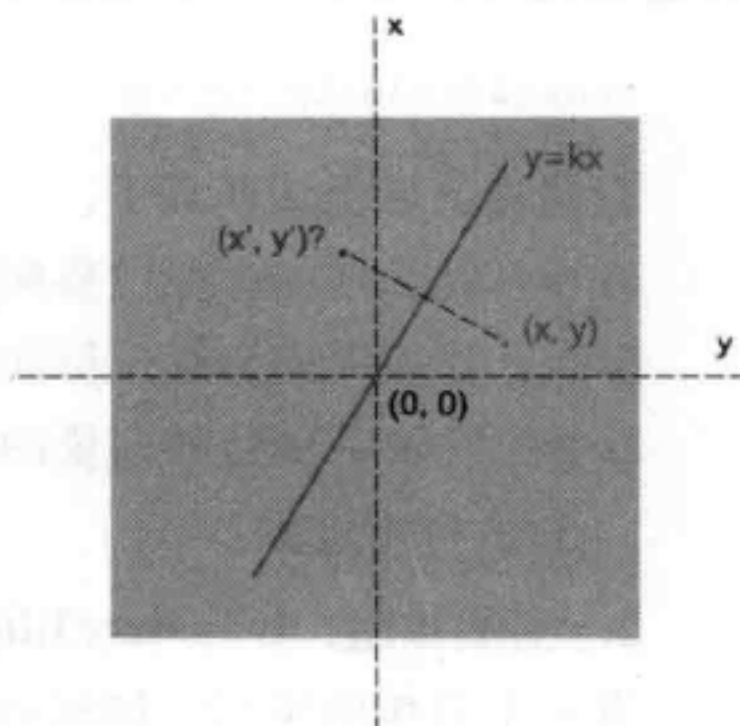


图 11-42 镜像坐标点 (x', y')

11.4 CSS3 3D 变形

使用二维变形能够改变元素在水平和垂直轴线,然而还有一个轴沿着它,可以改变元素。使用三维变形,可以改变元素在 Z 轴位置。

三维变换使用基于二维变换的相同属性,如果熟悉二维变换会发现,3D 变形的功能和 2D 变换的功能类似。CSS3 中的 3D 变换主要包括以下几种功能函数。

- 3D 位移: 包括 `translateZ()` 和 `translate3d()` 两个功能函数。
- 3D 旋转: 包括 `rotateX()`、`rotateY()`、`rotateZ()` 和 `rotate3d()` 四个功能函数。
- 3D 缩放: 包括 `scaleZ()` 和 `scale3d()` 两个功能函数。
- 3D 矩阵: 和 2D 变形一样,也有一个 3D 矩阵功能函数 `matrix3d()`。

11.4.1 3D 位移

CSS3 中 3D 位移主要包括两种函数 `translateZ()` 和 `translate3d()`。

1. `translate3d()` 函数的语法

`translate3d()` 函数使一个元素在三维空间移动。这种变形的特点是，使用三维向量的坐标定义元素在每个方向移动多少。其基本语法如下。

```
translate3d(tx,ty,tz)
```

其属性值取值说明如下。

- tx: 代表横向坐标位移向量的长度;
- ty: 代表纵向坐标位移向量的长度;
- tz: 代表 Z 轴位移向量的长度。此值不能是一个百分比值, 如果取值为百分比值, 将会认为无效值。

2. 实战体验: `translate3d()` 函数的效果

看一个简单的实例, 加深对 `translate3d()` 函数的理解。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>translate3d()</title>
  <style type="text/css">
    .stage {
      width: 300px;
      height: 300px;
      float: left;
      margin: 15px;
      position: relative;
      background: url(images/bg-grid.jpg) no-repeat center center;
      background-size: 100% 100%;

      perspective: 1200px;
    }
    .container {
      position: absolute;
      top: 50%;
      left: 50%;

      transform-style: preserve-3d;
    }
    .container img {
      position: absolute;
      margin-left: -35px;
      margin-top: -50px;
    }
  </style>
</head>
<body>
  <div class="stage">
    <div class="container">
      <img alt="A 3D cube with a grid background, demonstrating the translate3d() function." data-bbox="140 458 773 927"/>
    </div>
  </div>
</body>
</html>
```

```

.container img:nth-child(1){
    z-index: 1;
    opacity: .6;
}
.s1 img:nth-child(2){
    z-index: 2;
    transform: translate3d(30px, 30px, 200px);
}
.s2 img:nth-child(2){
    z-index: 2;
    transform: translate3d(30px, 30px, -200px);
}
...

```

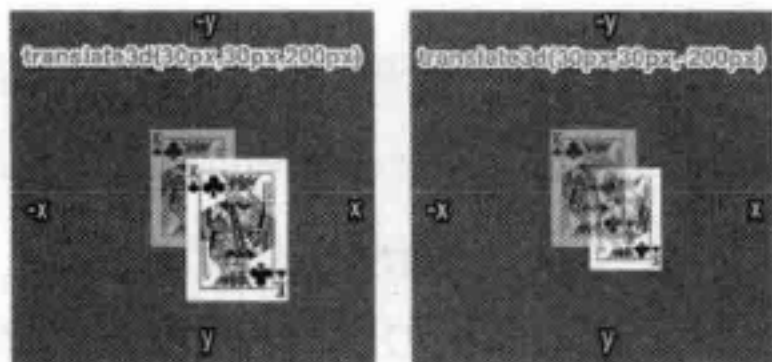


图 11-43 translate3d() 的效果

效果如图 11-43 所示。

从图 11-43 的效果可以看出，当 Z 轴值越大时，元素也离观看者更近，从视觉上元素就變得更大；反之其值越小时，元素也离观看者更远，从视觉上元素就变得更小。

3. translateZ() 函数的语法

CSS3 中 3D 位移还有 translateZ() 函数，功能是让元素在 3D 空间沿 Z 轴进行位移，其基本使用语法如下。

```
translate(t)
```

取值 t 指的是 Z 轴的向量位移长度。使用 translateZ() 函数可以让元素在 Z 轴进行位移，当其值为负值时，元素在 Z 轴越移越远，导致元素变得较小。反之，当其值为正值时，元素在 Z 轴越移越近，导致元素变得较大。

4. 实战体验：translateZ() 函数的效果

在上例的基础上，稍加变化一下，将 translate3d() 函数换成 translateZ() 函数。

```

.s1 img:nth-child(2){
    z-index: 2;
    opacity: .6;
    transform: translateZ(200px);
}
.s2 img:nth-child(2){
    z-index: 2;
    transform: translateZ(-200px);
}

```

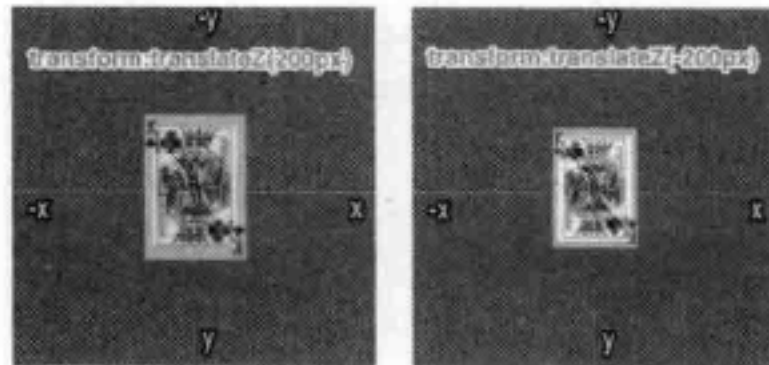


图 11-44 translate() 函数的效果

效果如图 11-44 所示。

图 11-44 的效果再次证明了 translateZ() 函数仅让元素在 Z 轴进行位移，当其值越大时，元素离观看者越近，视觉上元素放大，反之元素缩小。

translateZ() 函数在实际使用中等同于 translate3d(0,0,tz)。仅从视觉效果上看，translateZ() 和 translate3d(0,0,tz) 函数功能非常类似于二维空间的 scale() 缩放函数，但实际

上完全不同。translateZ() 和 translate3d(0,0,tz) 变形是发生在 Z 轴上，而不是 X 轴和 Y 轴。当使用 3D 变形，能够在 Z 轴上移动一个元素确实有很大的好处，比如说在创建一个 3D 立方体的盒子之时。

11.4.2 3D 缩放

CSS3 3D 变形中的缩放主要有 scaleZ() 和 scale3d() 两种函数，当 scale3d() 中 X 轴和 Y 轴同时为 1，即 scale3d(1,1,sz)，其效果等同于 scaleZ(sz)。

1. 3D 缩放函数的语法

通过使用 3D 缩放函数，可以让元素在 Z 轴上按比例缩放。默认值为 1，当值大于 1 时，元素放大，小于 1 大于 0.01 时，元素缩小。其使用语法如下。

```
scale3d(sx,sy,sz)
```

sx: 横向缩放比例; sy: 纵向缩放比例; sz: Z 轴缩放比例。

```
scaleZ(s)
```

取值 s 指定元素每个点在 Z 轴的比例。scaleZ(-1) 定义了一个原点在 Z 轴的对称点（按照元素的变换原点）。



提示 scaleZ() 和 scale3d() 函数单独使用时没有任何效果，需要配合其他的变形函数一起使用才会有效果。

2. 实战体验：按比例缩放纸牌

下面来看一个实例，为了能看到 scaleZ() 函数的效果，添加了 rotateX(45deg) 功能。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>scaleZ()</title>
  <style type="text/css">
    .stage {
      width: 300px;
      height: 300px;
      float: left;
      margin: 15px;
      position: relative;
      background: url(images/bg-grid.jpg) no-repeat center center;
      background-size: 100% 100%;

      perspective: 1200px;
    }
    .container {
```

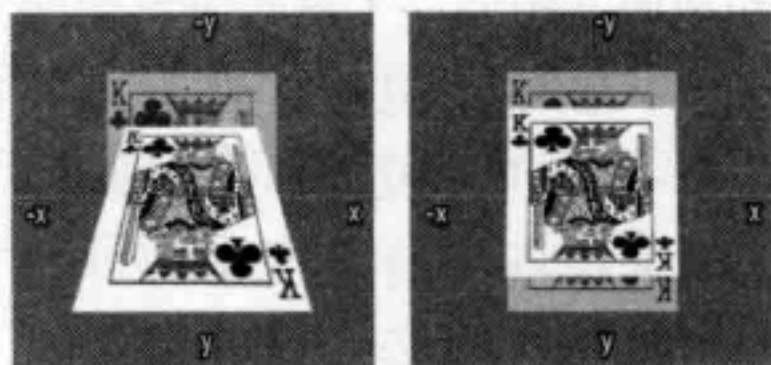


```

position: absolute;
top: 50%;
left: 50%;

transform-style: preserve-3d;
}
.container img {
position: absolute;
margin-left: -70px;
margin-top: -100px;
}
.container img:nth-child(1){
z-index: 1;
opacity: .6;
}
.s1 img:nth-child(2){
z-index: 2;
transform: scaleZ(5) rotateX(45deg);
}
.s2 img:nth-child(2){
z-index: 2;
transform: scaleZ(.25) rotateX(45deg);
}
...

```



效果如图 11-45 所示。

图 11-45 scaleZ() 函数效果

11.4.3 3D 旋转

到目前为止，已经讨论了如何让一个元素在平面上进行顺时针或逆时针旋转。在三维变形中，可以让元素在任何轴旋转。为此，CSS3 新增三个旋转函数 rotateX()、rotateY() 和 rotateZ()。接下来简单了解一下这三个旋转函数。

1. 3D 旋转函数的语法

在三维空间里，使用 rotateX()、rotateY() 和 rotateZ() 函数让一个元素围绕 X、Y、Z 轴旋转，其基本语法如下。

```

rotateX(a)
rotateY(a)
rotateZ(a)

```

其中 a 指的是一个旋转角度值，其值可以是正值也可以是负值。如果值为正值，元素顺时针旋转；反之，值为负，元素围绕逆时针旋转。



提示 rotateZ() 函数指定元素围绕 Z 轴旋转，如果仅从视觉角度上看，rotateZ() 函数让元素顺时针或逆时针旋转，并且效果和 rotate() 效果等同，但不是在 2D 平面旋转。

在三维空间里，除了 rotateX()、rotateY() 和 rotateZ() 函数可以让一个元素在三维空间中旋转之外，还有一个属性函数 rotate3d()。在 3D 空间，旋转有三个角度来描述一个转动轴。轴的旋转是由一个 [x,y,z] 向量并经过元素原点。其基本语法如下。

```
rotate3d(x,y,z,a)
```

rotate3d() 中取值说明。

- x: 0 ~ 1 的数值，用来描述元素围绕 X 轴旋转的矢量值。
- y: 0 ~ 1 的数值，用来描述元素围绕 Y 轴旋转的矢量值。
- z: 0 ~ 1 的数值，用来描述元素围绕 Z 轴旋转的矢量值。
- a: 角度值，用来指定元素在 3D 空间旋转的角度，如果其值为正值，元素顺时针旋转，反之元素逆时针旋转。

当 x、y、z 三个值同时为 0 时，元素在 3D 空间不做任何旋转。当 x、y、z 取不同值时，和前面介绍的三个旋转函数功能等同。

- rotateX(a) 函数功能等同于 rotate3d(1,0,0,a)。
- rotateY(a) 函数功能等同于 rotate3d(0,1,0,a)。
- rotateZ(a) 函数功能等同于 rotate3d(0,0,1,a)。

2. 实战体验：3D 旋转纸牌

接下来通过一个简单的实例，来说明 3D 旋转的运用。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>CSS3 3D 旋转</title>
  <style type="text/css">
    .stage {
      width: 300px;
      height: 300px;
      float: left;
      margin: 15px;
      position: relative;
      background: url(images/bg-grid.jpg) no-repeat center center;
      background-size: 100% 100%;
      perspective: 1200px;
    }
    .container {
      position: absolute;
      top: 50%;
      left: 50%;
      transform-style: preserve-3d;
    }
  </style>
</head>
<body>
```

```

.container img {
    position: absolute;
    margin-left: -70px;
    margin-top: -100px;
}
.container img:nth-child(1){
    z-index: 1;
    opacity: .6;
}
.s1 img:nth-child(2){
    z-index: 2;
    transform: rotateX(45deg);
}
.s2 img:nth-child(2){
    z-index: 2;
    transform: rotateY(45deg);
}
.s3 img:nth-child(2){
    z-index: 2;
    transform: rotateZ(45deg);
}
.s4 img:nth-child(2){
    z-index: 2;
    transform: rotate3d(.6, 1, .6, 45deg);
}
</style>
</head>
<body>
<div class="stage s1">
    <div class="container">
        
        
    </div>
</div>
<!-- S1 ~ S4 结构与 S1 相同 -->
</body>
</html>

```

效果如图 11-46 所示。

11.4.4 3D 矩阵

CSS3 中的 3D 矩阵比 2D 矩阵复杂, 从二维到三维, 是从 4 到 9; 而在矩阵里是 3×3 变成 4×4 , 即 9 到 16。对于 3D 矩阵而言, 本质上很多东西与 2D 一致, 只是复杂程度不一样而已。

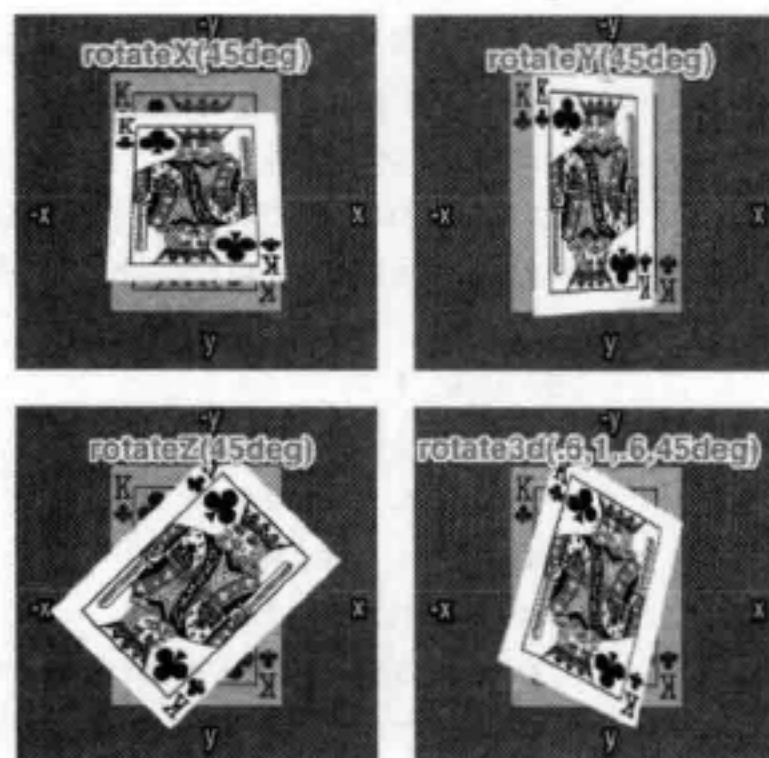


图 11-46 CSS3 3D 旋转效果


```

.container {
    position: relative;
    height: 230px;
    width: 100px;
    top: 50%;
    left: 50%;
    margin: -100px 0 0 -50px;
    transform-style: preserve-3d;
}
.container:hover{
    animation:spin 5s linear infinite;
}
.side {
    font-size: 20px;
    font-weight: bold;
    height: 100px;
    line-height: 100px;
    color: #fff;
    position: absolute;
    text-align: center;
    text-shadow: 0 -1px 0 rgba(0,0,0,0.2);
    text-transform: uppercase;
    width: 100px;
}
.top {
    background: #9acc53;
    transform: rotate(-45deg) skew(15deg, 15deg);
}
.left {
    background: #8ec63f;
    transform: rotate(15deg) skew(15deg, 15deg) translate(-50%, 100%);
}
.right {
    background: #80b239;
    transform: rotate(-15deg) skew(-15deg, -15deg) translate(50%, 100%);
}
</style>
</head>
<body>
<div class="stage s1">
    <div class="container">
        <div class="side top">1</div>
        <div class="side left">2</div>
        <div class="side right">3</div>
    </div>
</div>
</body>
</html>

```

效果如图 11-47 所示。

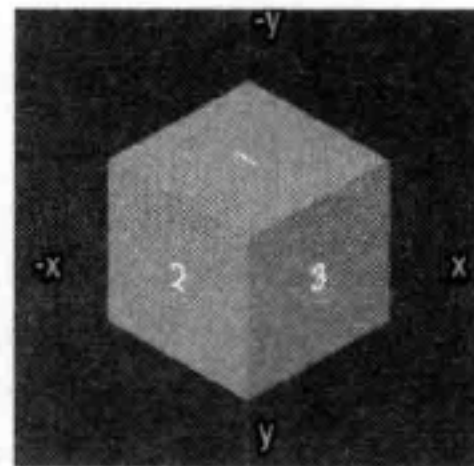


图 11-47 CSS3 2D 多重变换制作 3D 立方体

11.5.2 3D 多重变形制作立方体

上例通过三个面，使用多个 2D 变形制作一个 3D 立方体，接着使用 3D 多重变形制作一个 3D 立方体。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>CSS3 3D 多重变形制作 3D 立方体 </title>
  <style type="text/css">
    @keyframes spin{
      0%{transform:rotateY(0deg)}
      100%{transform:rotateY(360deg)}
    }
    .stage {
      width: 300px;
      height: 300px;
      margin: 15px auto;
      position: relative;
      background: url(images/bg-grid.jpg) no-repeat center center;
      background-size: 100% 100%;
      perspective: 300px;
    }
    .container {
      top: 50%;
      left: 50%;
      margin: -100px 0 0 -100px;
      position: absolute;
      transform: translateZ(-100px);
      transform-style: preserve-3d;
    }
    .container:hover{
      animation:spin 5s linear infinite;
    }
    .side {
      background: rgba(142,198,63,0.3);
      border: 2px solid #8ec63f;
      font-size: 60px;
      font-weight: bold;
      color: #fff;
      height: 196px;
      line-height: 196px;
      position: absolute;
      text-align: center;
      text-shadow: 0 -1px 0 rgba(0,0,0,0.2);
      text-transform: uppercase;
      width: 196px;
    }
```

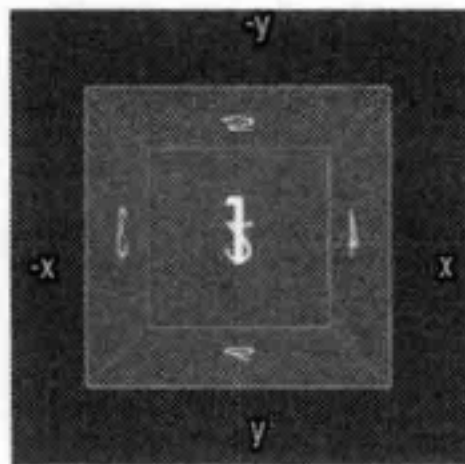


```

    }
    .front {
        transform: translateZ(100px);
    }
    .back {
        transform: rotateX(180deg) translateZ(100px);
    }
    .left {
        transform: rotateY(-90deg) translateZ(100px);
    }
    .right {
        transform: rotateY(90deg) translateZ(100px);
    }
    .top {
        transform: rotateX(90deg) translateZ(100px);
    }
    .bottom {
        transform: rotateX(-90deg) translateZ(100px);
    }
</style>
</head>
<body>

<div class="stage">
    <div class="container">
        <div class="side front">1</div>
        <div class="side back">2</div>
        <div class="side left">3</div>
        <div class="side right">4</div>
        <div class="side top">5</div>
        <div class="side bottom">6</div>
    </div>
</div>
</body>
</html>

```



效果如图 11-48 所示。

图 11-48 CSS3 3D 多重变形制作 3D 立方体

11.6 综合案例：3D 变形制作产品信息展示

在 Web 设计中有很多方法制作产品信息展示，例如鼠标移动到产品图片上时，产品信息滑动出来甚至使用弹出框。本节使用 CSS3 3D 变形制作一个 3D 立方体翻转展示信息的效果。

要实现的效果如图 11-49 所示。默认情况下只显示产品图片，而产品信息隐藏不可见。用户鼠标悬浮在产品图像上时，产品图像慢慢往上旋转使产品信息展示出来，而产品图像慢慢隐藏起来，看起来就像是一个旋转的立方体。



图 11-49 产品信息翻转的动画过程

该技术用于创建一个多维的数据，在这个实例中使用了两个元素用于正面和反面。前面用来放置产品图片，底部用来放置产品信息。默认情况下产品信息隐藏起来，同时鼠标悬停在产品图片上时，隐藏在底部的产品信息在 X 轴放置 -90° 度和 Z 轴旋转，使底部的信息旋转置于顶部，从而达到需要的效果，产品图片隐藏，产品信息显示。如图 11-50 所示。

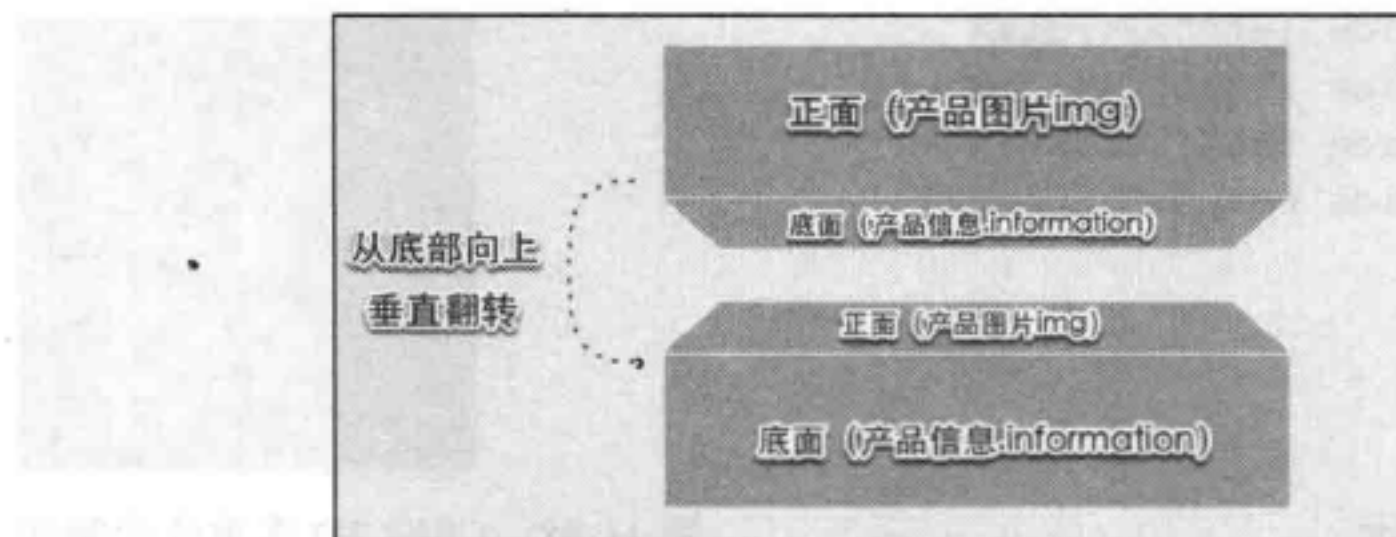


图 11-50 产品信息翻转过程

使用两个标签包裹产品图片和产品信息，第一个主要用来设置 3D 视点 perspective，旨在设置用户有画布的视距；第二个包裹容器主要用来包裹图片和产品信息。当鼠标悬浮在这个容器上时，会沿 X 轴旋转，将产品信息显示出来。在三维旋转中，常用的一种结构如下。

```
舞台 =>div.wrapper
容器 =>div.item
产品图片 =>img
产品信息 =>span.information
```

在实例中使用的结构如下。

```

<div class="wrapper">
  <div class="item">
    
    <span class="information">
      <strong>Contact Form</strong> The easiest way to add a contact form to your shop.
    </span>
  </div>
</div>

```

接下来完成这个实例的样式。首先给每个 wrapper 容器设置 `display:inline-block` 和 `perspective: 4000px`, 同时给 item 设置 `transform-style:preserve-3d` 让它的子元素具有一个 3D 位置。为了效果更加完美, 添加了 CSS3 的 `transition` 属性 (第 12 章详细介绍)。

```

.wrapper {
  display: inline-block;
  width: 310px;
  height: 100px;
  vertical-align: top;
  margin: 1em 1.5em 2em 0;
  cursor: pointer;
  position: relative;
  font-family: Tahoma, Arial;
  perspective: 4000px;
}

.item {
  height: 100px;
  transform-style: preserve-3d;
  transition: transform .6s;
}

```

接下来给产品图片设置一个 Z 轴位移, 给产品信息设置一个 X 轴旋转和 Z 轴位移, 为了效果更加完美, 还添加了一些其他的装饰样式。

```

.item img {
  display: block;
  position: absolute;
  top: 0;
  border-radius: 3px;
  box-shadow: 0px 3px 8px rgba(0,0,0,0.3);
  transform: translateZ(50px);
  transition: all .6s;
}

.item .information {
  display: block;
  position: absolute;
  top: 0;
  height: 80px;
  width: 290px;
}

```



```

text-align: left;
border-radius: 15px;
padding: 10px;
font-size: 12px;
text-shadow: 1px 1px 1px rgba(255,255,255,0.5);
box-shadow: none;
background: rgb(236,241,244);
background: linear-gradient(to bottom,
    rgba(236,241,244,1) 0%,rgba(190,202,217,1) 100%);
transform: rotateX(-90deg) translateZ(50px);
transition: all .6s;
}

```

最后实现用户鼠标悬停在产品图片上时，旋转隐藏产品图片，同时产品信息旋转显示出来。

```

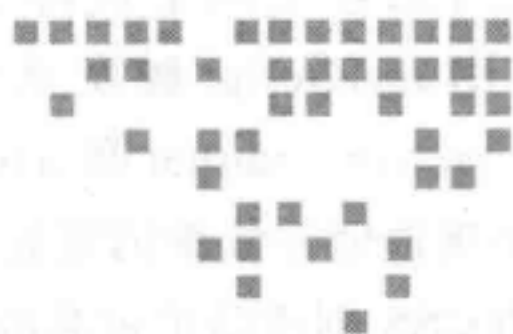
.item:hover {
    transform: translateZ(-50px) rotateX(95deg);
}
.item:hover img {
    box-shadow: none;
    border-radius: 15px;
}
.item:hover .information {
    box-shadow: 0px 3px 8px rgba(0,0,0,0.3);
    border-radius: 3px;
}

```

至此，整个实例的效果就全部完成了。

11.7 本章小结

本章主要分两个部分，第一部分是二维空间的变形，第二部分是三维空间的变形。二维空间中的变形主要介绍了旋转 `rotate()`、倾斜 `skew()`、位移 `translate()`、缩放 `scale()` 和 2D 矩阵等函数功能。三维空间的变形主要介绍了 3D 空间的几个重要概念，比如 `perspective`、`perspective-origin` 和 `backface-visibility` 等，以及详细介绍了 3D 空间中的 3D 旋转、3D 位移、3D 缩放和 3D 矩阵。通过本章的学习，大家对 CSS3 的变形会有较深的了解。



CSS3 过渡

多年来，Web 前端开发人员一直在寻求通过 HTML 和 CSS 实现一些动画交互效果，而不再使用 JavaScript 或 Flash。现在他们的愿望实现了。CSS3 除了给我们带来前面介绍的一些特殊功能模块之外，还为 Web 设计师添加了一些动画功能模块。

可以通过 `:hover`、`:focus`、`:active`、`:checked` 或者 JavaScript 触发一个元素，这样，外观变化会显得更细腻，而不会让人感觉“一触即发”。例如悬浮修改链接色，在 CSS2.1 中鼠标悬浮时，立刻从一个颜色变成另一个颜色。而在 CSS3 中使用过渡功能，鼠标悬浮时，颜色在一定的时间内，从一个颜色过渡到另一个颜色，给用户更好、更细腻的体验。

本章主要介绍如何使用 CSS3 的过渡属性，并帮助读者进一步掌握 CSS3 中 `transition` 的使用方法，实现属性值的开始值与属性值的结束值之间平滑过渡的动画。

12.1 CSS3 过渡简介

CSS3 的过渡功能像是一种黄油，通过一些 CSS 的简单动作触发样式平滑过渡。

W3C 标准中描述的 `transition` 功能很简单：CSS3 的 `transition` 允许 CSS 的属性值在一定的时间区间内平滑地过渡。这种效果可以在鼠标单击、获得焦点、被点击或对元素任何改变中触发，并平滑地以动画效果改变 CSS 的属性值。

12.1.1 如何创建简单的过渡

以往 Web 中的动画都是依赖于 JavaScript 和 Flash 实现，但原生 CSS 过渡在客户端上需要处理的资源少得多，从而显得更加平滑。

CSS3 过渡与元素上的常规样式一起声明。只要目标属性更改，浏览器就会应用过渡。除了使用 JavaScript 触发动作外，在 CSS 中也可以通过一些伪类来触发，如：hover、:focus、:active、:target 和 :checked 等。这很重要：无须在 JavaScript 中编写动画，只需要更改一个属性值并依赖浏览器来执行所有重要工作。






以下是使用 CSS 创建简单过渡的步骤。

- 1) 在默认样式中声明元素的初始状态样式。
- 2) 声明过渡元素最终状态样式，比如悬浮状态。
- 3) 在默认样式中通过添加过渡函数，添加一些不同的样式。

12.1.2 浏览器兼容性

transition 属性和 CSS3 其他的属性一样，离不开浏览器对它的束缚，在实际使用的时候都需要使用各浏览器厂商的私有属性。不过有个令人兴奋的消息，到目前为止，在 IE 10+、Firefox 16+、Chrome 26+、Safari 7+ 和 Opera 15.0+ 等浏览器中可以支持 transition 的标准属性。具体的浏览器兼容性参见表 12-1。

表 12-1 transition 浏览器兼容性

属性					
transition	10+ ✓	4.0+ ✓	3.1+ ✓	4.0+ ✓	10.5+ ✓

虽然主流浏览器对 transition 属性的支持可以说是很完美了，但并不能代表所有用户都时刻关注与更新自己使用的浏览器。想让效果更满意，很有必要花点时间详细了解兼容性。

- ❑ IE 10+(PP3)、Firefox 4.0 ~ 15.0、Chrome 4.0 ~ 20.0、Safari 3.1 ~ 6.0 和 Opera 10.5 ~ 12.0，在这些浏览器中，使用 transition 属性都需要添加各浏览器的私有属性。
- ❑ IE 10+、Firefox 16.0+、Chrome 26.0+、Safari 7.0+、Opera 12.1+ 支持 transition 属性的标准语法，不需要添加浏览器的私有属性。
- ❑ iOS Safari 3.2 ~ 6.1、Android Browser 2.1+、Blackberry Browser 7.0+ 和 Chrome for Android 27.0 需要添加浏览器前缀 -webkit-，Opera Mobile 10.0 ~ 12.0 中需要添加浏览器前缀 -o-。
- ❑ iOS Safari 7.0+ 和 Firefox for Android 22.0+ 支持 transition 属性的标准语法，不需要添加浏览器的私有属性。

12.1.3 CSS3 过渡属性

transition 属性是一个复合属性，出于简洁性和便于维护考虑，过渡语法通常以简化的形式表达。

```
transition : [<'transition-property'> || <'transition-duration'> ||
```



```
<'transition-timing-function'> || <'transition-delay'> [,
[<'transition-property'> || <'transition-duration'> ||
<'transition-timing-function'> || <'transition-delay'>]]*
```

transition 属性主要包含四个属性值。

□ transition-property: 指定过渡或动态模拟的 CSS 属性。

□ transition-duration: 指定完成过渡所需的时间。

□ transition-timing-function: 指定过渡函数。

□ transition-delay: 指定过渡开始出现的延迟时间。

如果单独声明 transition 属性会变得非常的密集,特别是在添加各个浏览器的私有前缀的时候。值得庆幸的是,transition 属性可以像 border、margin、padding 和 font 这样的属性,将上面介绍的 transition 的四个子属性 (transition-property、transition-duration、transition-timing-function 和 transition-delay) 简写在一起。需要注意的是,这四个子属性之间不能使用逗号分隔,而是使用空格分隔。transition 属性的简写语法如下。

```
transition:<single-transition>[',' <single-transition>*]
```

其中 <single-transition> 等于:

```
single-transition= [transition-property] || [transition-duration] ||
[transition-timing-function] || [transition-delay]
```

值得注意的一点是,在 transition 属性简写中,transition-duration 和 transition-delay 取值都是 <time>,所以在简写中要区分它们。一般浏览器会根据先后顺序决定:第一个 <time> 会解析为 transition-duration,而第二个 <time> 会解析为 transition-delay。

例如上面的实例,可以按 transition 简写的语法规则如下形式:

```
.transition{
  background: #8ec63f;
  width: 100px;
  height: 100px;
  border-radius: .5em;
  -webkit-transition: background 2s linear 2s, border-radius 3s ease-in 4s;
  -moz-transition: background 2s linear 2s, border-radius 3s ease-in 4s;
  -o-transition: background 2s linear 2s, border-radius 3s ease-in 4s;
  -ms-transition: background 2s linear 2s, border-radius 3s ease-in 4s;
  transition: background 2s linear 2s, border-radius 3s ease-in 4s;
}
.transition:hover {
  background:#f7941d;
  border-radius: 50%;
}
```

综合上述,可以给 transition 一个速记法。

```
transition:<property> <duration> <animation type> <delay>
```

也就是：

transition:< 过渡属性 > < 过渡所需时间 > < 过渡动画函数 > < 过渡延迟时间 >

transition 属性缩写规则如图 12-1 所示。

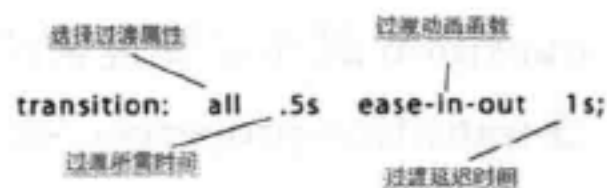


图 12-1 transition 属性缩写规则

12.2 CSS3 过渡子属性详解

在 transition 属性中包括四个子属性：transition-property、transition-duration、transition-timing-function 和 transition-delay。在使用 transition 时不是每个子属性都使用，其中前三个子属性是最受欢迎的。

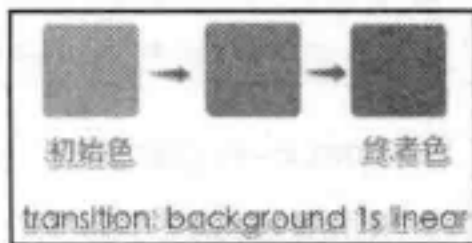
在下面的例子中，鼠标悬停在盒子上时，其背景色在 1 秒内以线性的方式从“#8ec63f”色过渡到“#f7941d”色。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title> 简单的过渡使用 </title>
  <style type="text/css" media="screen">
    .transition{
      background: #8ec63f;
      width: 100px;
      height: 100px;
      border-radius: .5em;
      -webkit-transition-property: background;
      -webkit-transition-duration: 1s;
      -webkit-transition-timing-function: linear;
      -moz-transition-property: background;
      -moz-transition-duration: 1s;
      -moz-transition-timing-function: linear;
      -o-transition-property: background;
      -o-transition-duration: 1s;
      -o-transition-timing-function: linear;
      -ms-transition-property: background;
      -ms-transition-duration: 1s;
      -ms-transition-timing-function: linear;
      transition-property: background;
      transition-duration: 1s;
      transition-timing-function: linear;
    }
    .transition:hover {
      background:#f7941d;
    }
  </style>
</head>
<body>
```

```
<div class="transition"></div>
</body>
</html>
```

鼠标悬浮在盒子上时,盒子背景色慢慢过渡到另一个色,如图 12-2 所示。

先从简单的例子中体验一下 transition 的效果,接下来针对每个子属性做一定的介绍。



☆图 12-2 盒子背景色过渡

12.2.1 指定过渡属性 transition-property

要让 transition 属性能正常工作,需要给元素设置两套样式用于用户与页面的交互。在 transition 中主要通过子属性 transition-property 来完成。简单来说,就是通过 transition-property 属性来指定过渡动画的 CSS 属性名称。该属性的基本语法如下。

```
transition-property: none | all | <single-transition-property>
[',' <single-transition-property>] *
```

取值简单说明如下。

- none: 没有指定任何样式。
- all: 默认值,表示指定元素所有支持 transition-property 属性的样式。
- <single-transition-property>: 指定样式,其等于 all 或者 <IDENT>。

需要特别注意,用 transition-property 来指定过渡属性并不是所有属性都可以过渡,只有属性具有一个中点值的属性才能具备过渡效果,其对应的类型属性主要如下。

- 颜色属性: 通过红、绿、蓝和透明度组合的过渡(每个数值处理),如 background-color、border-color、color 和 outline-color 等 CSS 样式属性。
- 具有长度值(length)、百分比(percentage)的属性: 真实的数字,如 word-spacing、width、vertical-align、top、right、bottom、left、min-width、min-height、max-width、max-height、line-height、height、background-position 等。
- integer: 离散步骤(整个数字),在真实的数字空间,以及使用 floor() 转换为整数时发生,如 outline-offset、z-index 等。
- number 真实的(浮点型)数值: 如 zoom、opacity、font-weight 等。
- 变形系列属性: 如 rotate()、rotate3d()、scale()、scale3d()、skew()、translate()、translate3d() 等。
- rectangle: 通过 x、y、width 和 height (转为数值)变换,如 crop 属性。
- visibility: 离散步骤,在 0 ~ 1 范围内。0 表示隐藏,1 表示完全显示,如 visibility 属性。
- 阴影: 作用于 color、x、y 和 blur 属性,如 text-shadow 属性。
- 渐变 (gradient): 通过每次停止时的位置和颜色进行变化。它们必须有相同的类

型（放射状的或是线性的）和相同的停止数值以便执行动画，如 background-image 属性。

- ❑ paint server(SVG)：只支持下面的情况。从 gradient 到 gradient，以及 color 到 color，然后工作与上面类似。
- ❑ space-separated list of above：如果列表有相同的项目数值，则列表每一项按照上面的规则进行变化，否则无变化。
- ❑ 缩写属性：如果缩写的所有部分都可以实现动画，则会像所有单个属性的变化一样。

具体什么 CSS 属性可以实现 transition 效果，在 W3C 官网中列出了所有可以实现 transition 效果的 CSS 属性值以及值的类型，如表 12-2 所示。

表 12-2 支持 transition 过渡功能的 CSS 属性表

background-color	background-position	border-bottom-color	border-bottom-width
border-left-color	border-left-width	border-right-color	border-right-width
border-spacing	border-top-color	border-top-width	bottom
clip	color	font-size	font-weight
height	left	letter-spacing	line-height
margin-bottom	margin-left	margin-right	margin-top
max-height	max-width	min-height	min-width
opacity	outline-color	outline-width	padding-bottom
padding-left	padding-right	padding-top	right
text-indent	text-shadow	vertical-align	visibility
width	word-spacing	z-index	

在设置过渡属性时，不局限于单一的一套样式转换，可以同时设置多个过渡样式，只是在样式之间需要使用逗号隔开。基于上面的实例，来做一个变化。

当鼠标悬浮在盒子上时，盒子先从高度 100px 过渡到 50px，然后宽度从 50px 过渡到 300px，同时盒子背景色从 “#8ec63f” 色过渡到 “#f7941d” 色。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title> 多个过渡属性的过渡使用 </title>
  <style type="text/css" media="screen">
    .transition{
      background: #8ec63f;
      width: 100px;
      height: 100px;
      border-radius: .5em;
      -webkit-transition-property: background,height,width;
      -webkit-transition-duration: 1s,1s,1s;
      -webkit-transition-timing-function: linear,linear,linear;
      -webkit-transition-delay: 1s,0s,1s;
```

```

-moz-transition-property: background,height,width;
-moz-transition-duration: 1s,1s,1s;
-moz-transition-timing-function: linear,linear,linear;
-moz-transition-delay: 1s,0s,1s;
-o-transition-property: background,height,width;
-o-transition-duration: 1s,1s,1s;
-o-transition-timing-function: linear,linear,linear;
-o-transition-delay: 1s,0s,1s;
-ms-transition-property: background,height,width;
-ms-transition-duration: 1s,1s,1s;
-ms-transition-timing-function: linear,linear,linear;
-ms-transition-delay: 1s,0s,1s;
transition-property: background,height,width;
transition-duration: 1s,1s,1s;
transition-timing-function: linear,linear,linear;
transition-delay: 1s,0s,1s;
}
.transition:hover {
    background:#f7941d;
    width: 300px;
    height: 50px;
}
</style>
</head>
<body>
    <div class="transition"></div>
</body>
</html>

```



多属性过渡效果如图 12-3 所示。

☆图 12-3 多属性过渡效果

12.2.2 指定过渡所需时间 transition-duration

transition-duration 属性主要用来设置一个属性过渡到另一个属性所需的时间，即从旧属性过渡到新属性花费的时间。该属性的基本语法如下。

```
transition-duration: <time>[,<time>] *
```

<time> 为数值，单位为 s（秒）或 ms（毫秒）。可以作用于所有元素，包括 :before 和 :after 伪元素。其默认值为 0，也就是变换是即时的。换句话说，当 transition-duration 取值为 0 时，指定元素样式过渡时，看不到过渡过程，直接看到结果。

和 transition-property 属性一样，当设置多个过渡属性时，也可以设置多个 transition-duration，每个值之间同样使用逗号分隔。而且每个值按顺序对应 transition-property 的属性值。基于上例，稍做修改，将正方形过渡到圆形。

```

<!DOCTYPE HTML>
<html lang="en-US">

```

```

<head>
  <meta charset="UTF-8">
  <title>transition-duration 的基本应用 </title>
  <style type="text/css" media="screen">
    .transition{
      background: #8ec63f;
      width: 100px;
      height: 100px;
      border-radius: .5em;
      -webkit-transition-property: background,border-radius;
      -webkit-transition-duration: .5s,1s;
      -webkit-transition-timing-function: linear,ease-in;
      -webkit-transition-delay: 1s;
      -moz-transition-property: background,border-radius;
      -moz-transition-duration: .5s,1s;
      -moz-transition-timing-function: linear,ease-in;
      -moz-transition-delay: 1s;
      -o-transition-property: background,border-radius;
      -o-transition-duration: .5s,1s;
      -o-transition-timing-function: linear,ease-in;
      -o-transition-delay: 1s;
      -ms-transition-property: background,border-radius;
      -ms-transition-duration: .5s,1s;
      -ms-transition-timing-function: linear,ease-in;
      -ms-transition-delay: 1s;
      transition-property: background,border-radius;
      transition-duration: .5s,1s;
      transition-timing-function: linear,ease-in;
      transition-delay: 1s;
    }
    .transition:hover {
      background:#f7941d;
      border-radius: 50%;
    }
  </style>
</head>
<body>
  <div class="transition"></div>
</body>
</html>

```



其效果如图 12-4 所示。

☆图 12-4 transition-duration 效果



注意 transition 同时设置了多个过渡属性时，如果它们过渡所需的时间都一样，没有必要为每个属性都设置一个过渡所需时间 transition-duration，只需要给 transition-duration 设置一个值就行。这个值将会运用到所有的过渡属性上。

12.2.3 指定过渡函数 transition-timing-function

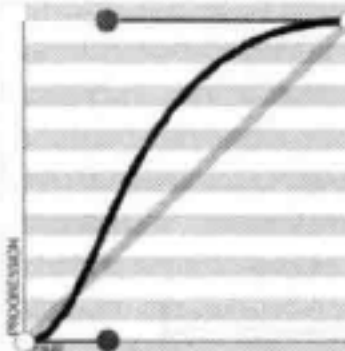
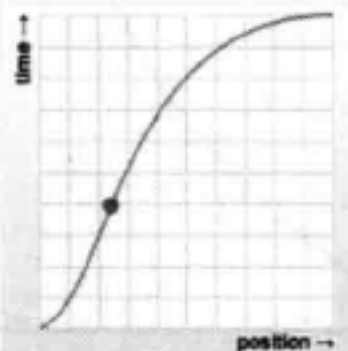
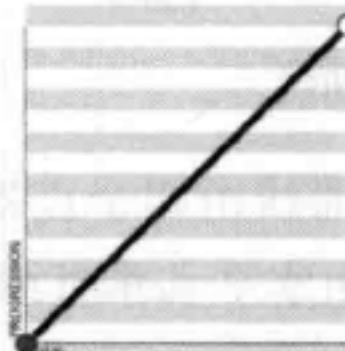
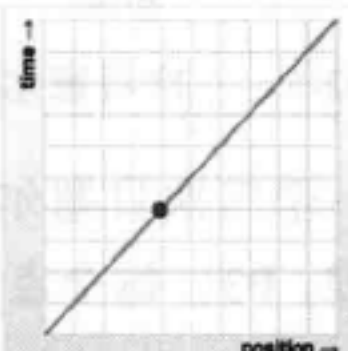
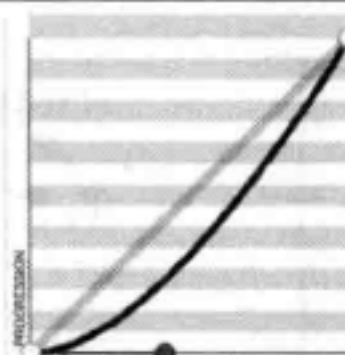
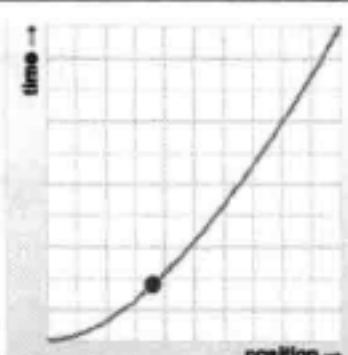

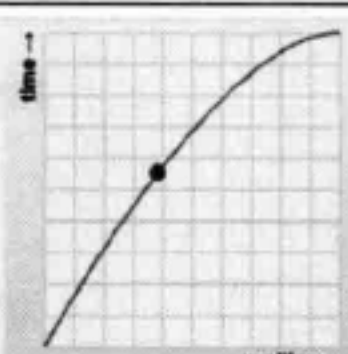
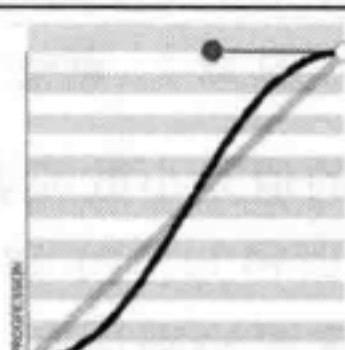
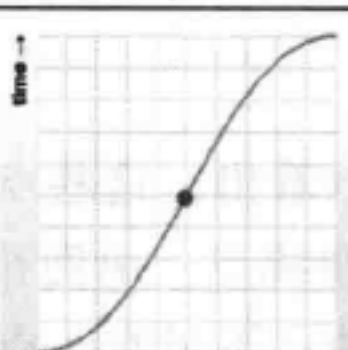
transition-timing-function 属性指定某种指代过渡“缓动函数”的属性。此属性可指定浏览器的过渡速度，以及过渡期间的操作进展情况，可以将某个值指定为预定义函数、阶梯函数或者三次贝塞尔曲线。该属性的基本语法如下。

```
transition-timing-function: <single-transition-timing-function>
['', '<single-transition-timing-function>']
```

1. 单一的过渡函数

<single-transition-timing-function> 指单一的过渡函数，主要包括表 12-3 所示的几个。

表 12-3 transition-timing-function 单一的过渡函数

函数	功能描述	图例																								
ease	默认值，元素样式从初始状态过渡到终止状态时速度由快到慢，逐渐变慢	  <table><thead><tr><th>Time (s)</th><th>Position</th></tr></thead><tbody><tr><td>0.000</td><td>0%</td></tr><tr><td>0.500</td><td>9%</td></tr><tr><td>1.000</td><td>29%</td></tr><tr><td>1.500</td><td>51%</td></tr><tr><td>2.000</td><td>69%</td></tr><tr><td>2.500</td><td>81%</td></tr><tr><td>3.000</td><td>89%</td></tr><tr><td>3.500</td><td>94%</td></tr><tr><td>4.000</td><td>98%</td></tr><tr><td>4.500</td><td>99%</td></tr><tr><td>5.000</td><td>100%</td></tr></tbody></table>	Time (s)	Position	0.000	0%	0.500	9%	1.000	29%	1.500	51%	2.000	69%	2.500	81%	3.000	89%	3.500	94%	4.000	98%	4.500	99%	5.000	100%
Time (s)	Position																									
0.000	0%																									
0.500	9%																									
1.000	29%																									
1.500	51%																									
2.000	69%																									
2.500	81%																									
3.000	89%																									
3.500	94%																									
4.000	98%																									
4.500	99%																									
5.000	100%																									
linear	元素样式从初始状态过渡到终止状态速度是恒速	  <table><thead><tr><th>Time (s)</th><th>Position</th></tr></thead><tbody><tr><td>0.000</td><td>0%</td></tr><tr><td>0.500</td><td>10%</td></tr><tr><td>1.000</td><td>20%</td></tr><tr><td>1.500</td><td>30%</td></tr><tr><td>2.000</td><td>40%</td></tr><tr><td>2.500</td><td>50%</td></tr><tr><td>3.000</td><td>60%</td></tr><tr><td>3.500</td><td>70%</td></tr><tr><td>4.000</td><td>80%</td></tr><tr><td>4.500</td><td>90%</td></tr><tr><td>5.000</td><td>100%</td></tr></tbody></table>	Time (s)	Position	0.000	0%	0.500	10%	1.000	20%	1.500	30%	2.000	40%	2.500	50%	3.000	60%	3.500	70%	4.000	80%	4.500	90%	5.000	100%
Time (s)	Position																									
0.000	0%																									
0.500	10%																									
1.000	20%																									
1.500	30%																									
2.000	40%																									
2.500	50%																									
3.000	60%																									
3.500	70%																									
4.000	80%																									
4.500	90%																									
5.000	100%																									
ease-in	元素样式从初始状态过渡到终止状态时，速度越来越快，呈一种加速状态。这种效果称为渐显效果	  <table><thead><tr><th>Time (s)</th><th>Position</th></tr></thead><tbody><tr><td>0.000</td><td>0%</td></tr><tr><td>0.500</td><td>2%</td></tr><tr><td>1.000</td><td>6%</td></tr><tr><td>1.500</td><td>13%</td></tr><tr><td>2.000</td><td>21%</td></tr><tr><td>2.500</td><td>32%</td></tr><tr><td>3.000</td><td>43%</td></tr><tr><td>3.500</td><td>55%</td></tr><tr><td>4.000</td><td>69%</td></tr><tr><td>4.500</td><td>84%</td></tr><tr><td>5.000</td><td>100%</td></tr></tbody></table>	Time (s)	Position	0.000	0%	0.500	2%	1.000	6%	1.500	13%	2.000	21%	2.500	32%	3.000	43%	3.500	55%	4.000	69%	4.500	84%	5.000	100%
Time (s)	Position																									
0.000	0%																									
0.500	2%																									
1.000	6%																									
1.500	13%																									
2.000	21%																									
2.500	32%																									
3.000	43%																									
3.500	55%																									
4.000	69%																									
4.500	84%																									
5.000	100%																									
ease-out	元素样式从初始状态过渡到终止状态时，速度越来越慢，呈一种减速状态。这种效果称为渐隐效果	  <table><thead><tr><th>Time (s)</th><th>Position</th></tr></thead><tbody><tr><td>0.000</td><td>0%</td></tr><tr><td>0.500</td><td>16%</td></tr><tr><td>1.000</td><td>31%</td></tr><tr><td>1.500</td><td>45%</td></tr><tr><td>2.000</td><td>57%</td></tr><tr><td>2.500</td><td>68%</td></tr><tr><td>3.000</td><td>79%</td></tr><tr><td>3.500</td><td>87%</td></tr><tr><td>4.000</td><td>94%</td></tr><tr><td>4.500</td><td>98%</td></tr><tr><td>5.000</td><td>100%</td></tr></tbody></table>	Time (s)	Position	0.000	0%	0.500	16%	1.000	31%	1.500	45%	2.000	57%	2.500	68%	3.000	79%	3.500	87%	4.000	94%	4.500	98%	5.000	100%
Time (s)	Position																									
0.000	0%																									
0.500	16%																									
1.000	31%																									
1.500	45%																									
2.000	57%																									
2.500	68%																									
3.000	79%																									
3.500	87%																									
4.000	94%																									
4.500	98%																									
5.000	100%																									
ease-in-out	元素样式从初始状态到终止状态时，先加速再减速。这种效果称为渐显渐隐效果	  <table><thead><tr><th>Time (s)</th><th>Position</th></tr></thead><tbody><tr><td>0.000</td><td>0%</td></tr><tr><td>0.500</td><td>2%</td></tr><tr><td>1.000</td><td>8%</td></tr><tr><td>1.500</td><td>19%</td></tr><tr><td>2.000</td><td>33%</td></tr><tr><td>2.500</td><td>50%</td></tr><tr><td>3.000</td><td>67%</td></tr><tr><td>3.500</td><td>81%</td></tr><tr><td>4.000</td><td>92%</td></tr><tr><td>4.500</td><td>98%</td></tr><tr><td>5.000</td><td>100%</td></tr></tbody></table>	Time (s)	Position	0.000	0%	0.500	2%	1.000	8%	1.500	19%	2.000	33%	2.500	50%	3.000	67%	3.500	81%	4.000	92%	4.500	98%	5.000	100%
Time (s)	Position																									
0.000	0%																									
0.500	2%																									
1.000	8%																									
1.500	19%																									
2.000	33%																									
2.500	50%																									
3.000	67%																									
3.500	81%																									
4.000	92%																									
4.500	98%																									
5.000	100%																									

2. 三次贝塞尔曲线

到目前为止，看到的 ease、linear、ease-in、ease-out 和 ease-in-out 等曲线函数非常一般，用于过渡动画中也不是十分精确。而现在制作一些动画需求越来越精确，需要定义一些更精确的函数。

先来看一个简单的三次贝塞尔曲线，如图 12-5 所示。

实际上，三次贝塞尔曲线有多个精确控制点，如图 12-6 所示。

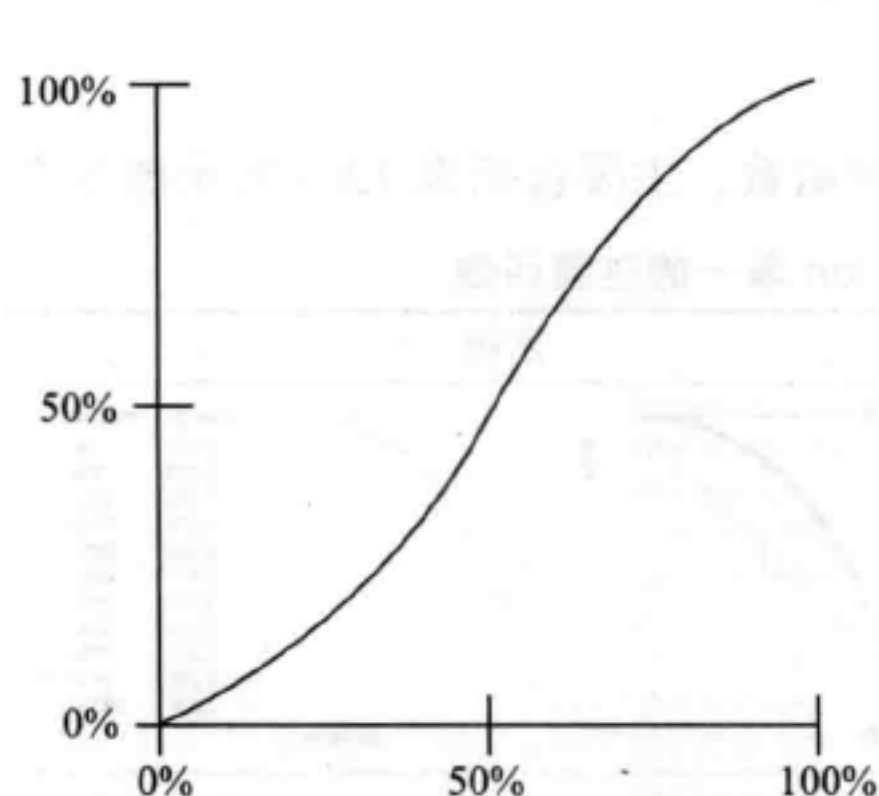


图 12-5 三次贝塞尔曲线

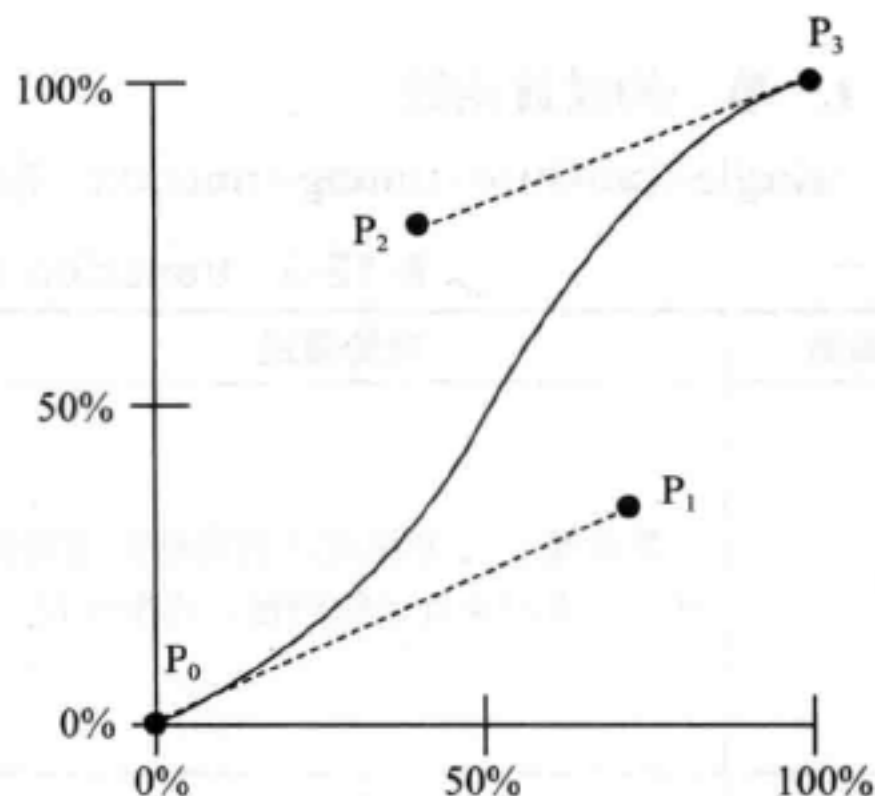


图 12-6 带有控制点的三次贝塞尔曲线

要知道，一个三次贝塞尔曲线由四个点控制曲线形状。图 12-7 所示曲线中标记的四个点为 P_0 , P_1 , P_2 , P_3 。每个点由水平和垂直两个值来确定，也就是常说的由 X 和 Y 值确定。

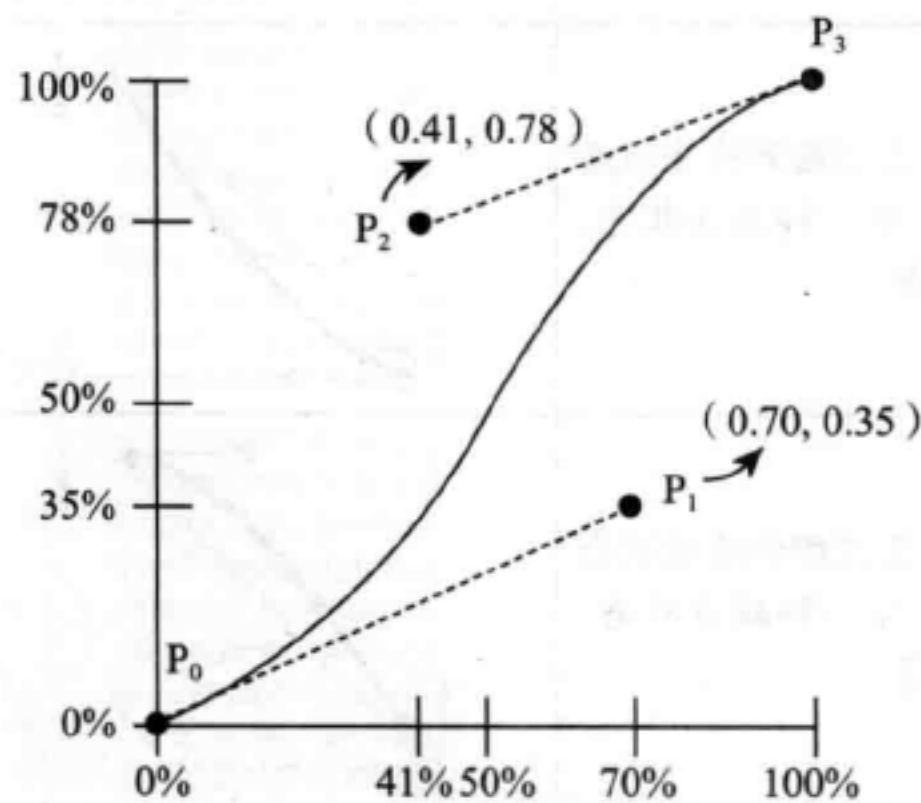


图 12-7 调整三次贝塞尔曲线

从上图可以看出，这些点的值是小数或者百分比，不过很少看到使用百分比来设置一个三次贝塞尔曲线点。由图可以得知， P_1 和 P_2 点的值很容易得到，而且它们的值都是一个 0 ~ 1 的小数。不过有一点需要特别注意，三次贝塞尔曲线中的 P_0 和 P_3 两个点是无法

设置的, 因为它们总是存在 HTML 中, 也就是说它们总会是 (0, 0) 和 (1, 1)。

仅上面的内容, 还无法完全的理解三次贝塞尔曲线。其实, 在计算机图形学中, 三次贝塞尔曲线是一种常用的模型。如果一直在使用制作编辑软件 (如 Photoshop), 会发现这个曲线模型, 可以使用钢笔工具来画, 这个曲线是由四个点来绘制, 如图 12-8 所示。

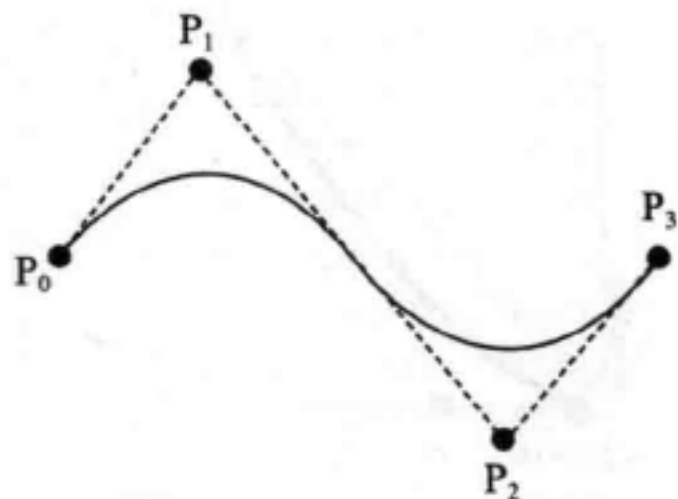


图 12-8 Photoshop 的钢笔工具绘制三次贝塞尔曲线

CSS3 的 transition 中, 定义三次贝塞尔曲线的语法如下。

```
cubic-bezier(P0,P1,P2,P3)
```

注意, 三次贝塞尔曲线中的每个点值只允许 0 ~ 1 的值。回到过渡函数 transition-timing-function 中, 看看它们是如何真正形成三次贝塞尔曲线的。

```
transition-timing-function: cubic-bezier(.85,0,1,1)
```

实现的曲线效果和过渡效果如图 12-9 所示。

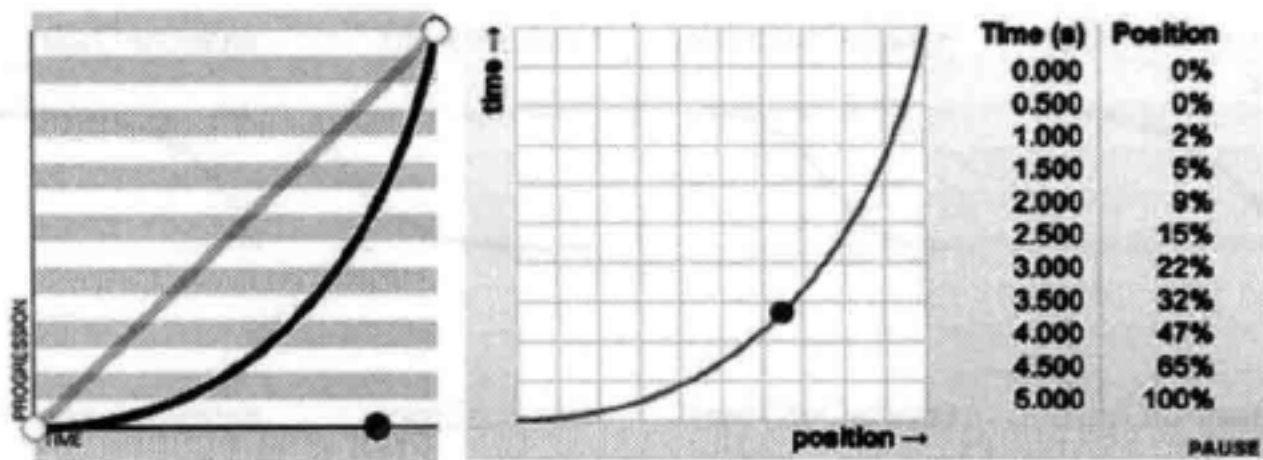


图 12-9 transition-timing-function 取值为 cubic-bezier (.85, 0, 1, 1) 的运动曲线

前面提到的 transition-timing-function 函数 ease、linear、ease-in、ease-out 和 ease-in-out 曲线函数都可以使用三次贝塞尔曲线实现, 如图 12-10 所示。

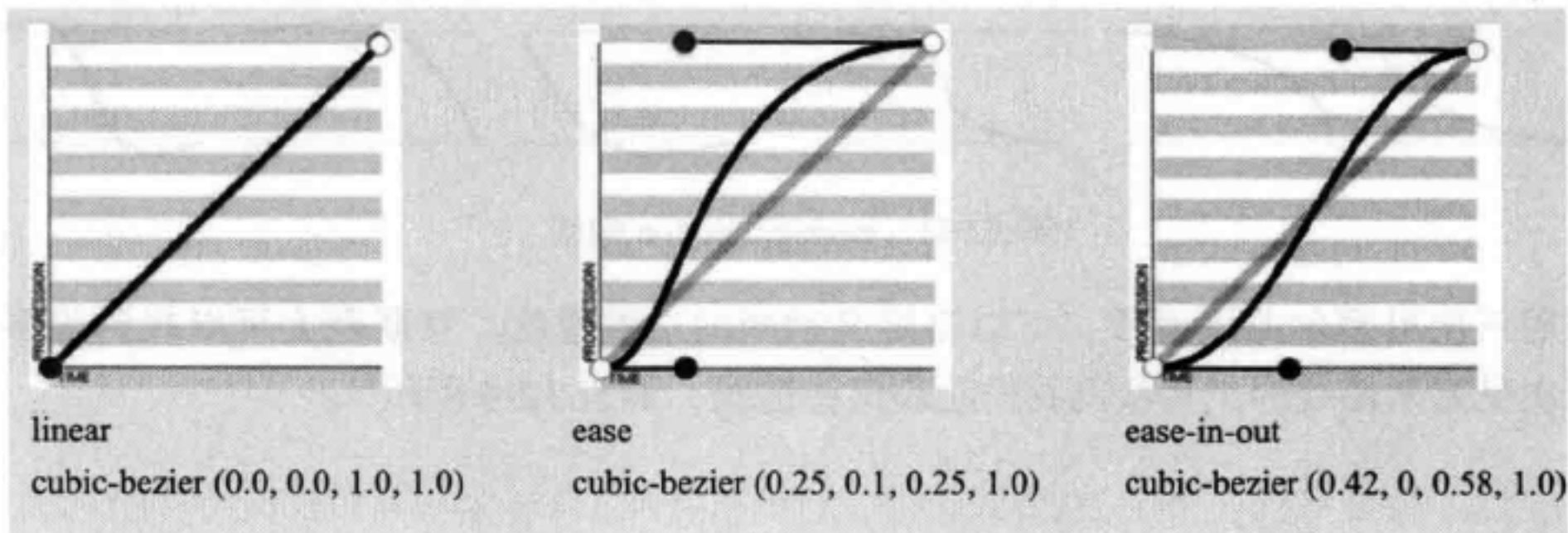


图 12-10 linear、ease、ease-in、ease-out、ease-in-out 对应的三次贝塞尔曲线

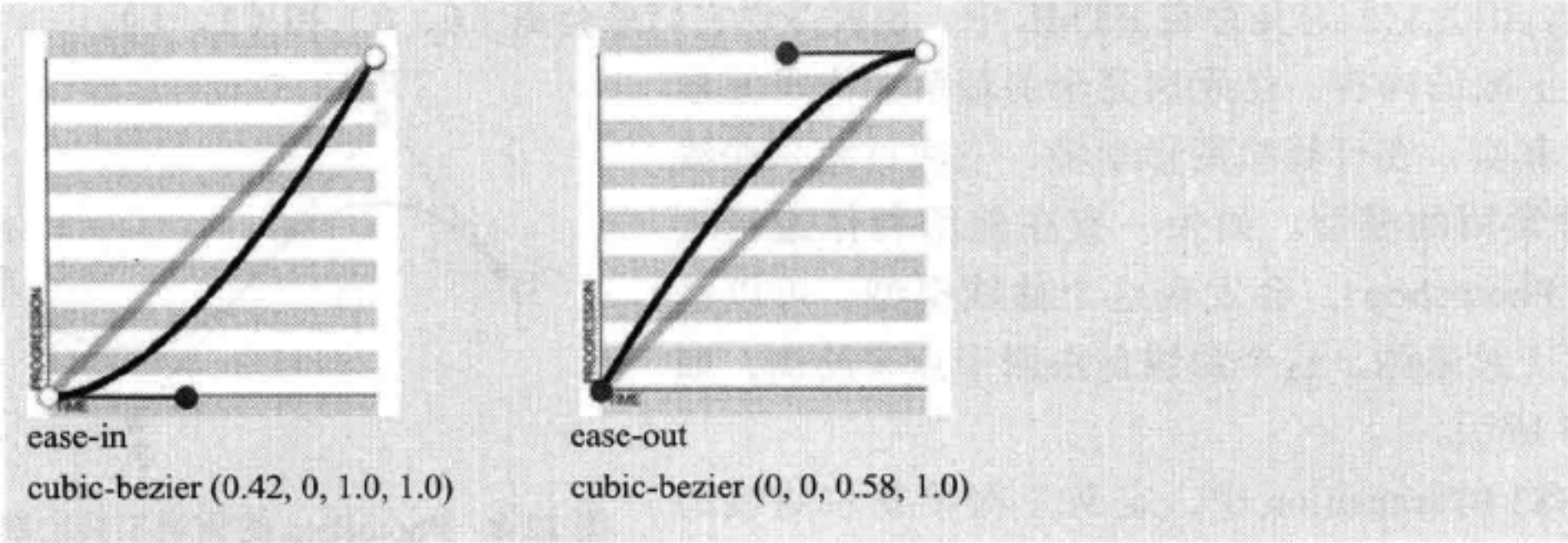


图 12-10 (续)

也可以使用 cubic-bezier 绘制类似于 easing function 函数的曲线，如图 12-11 所示。

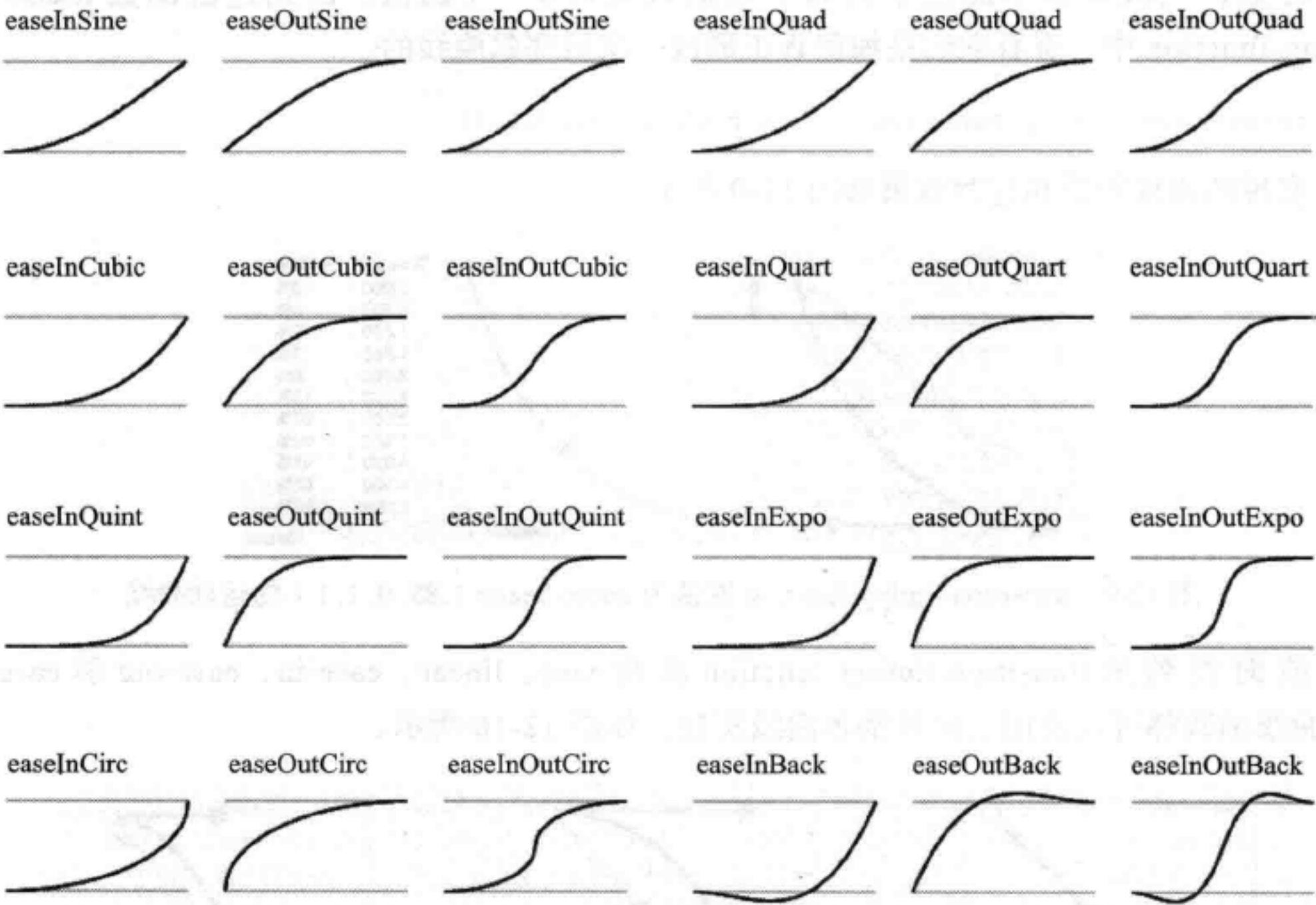
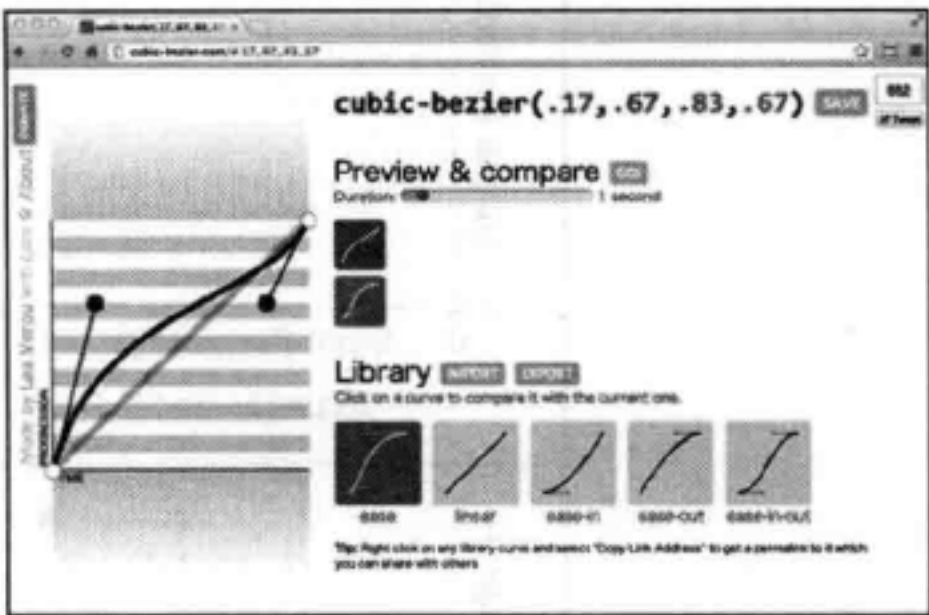
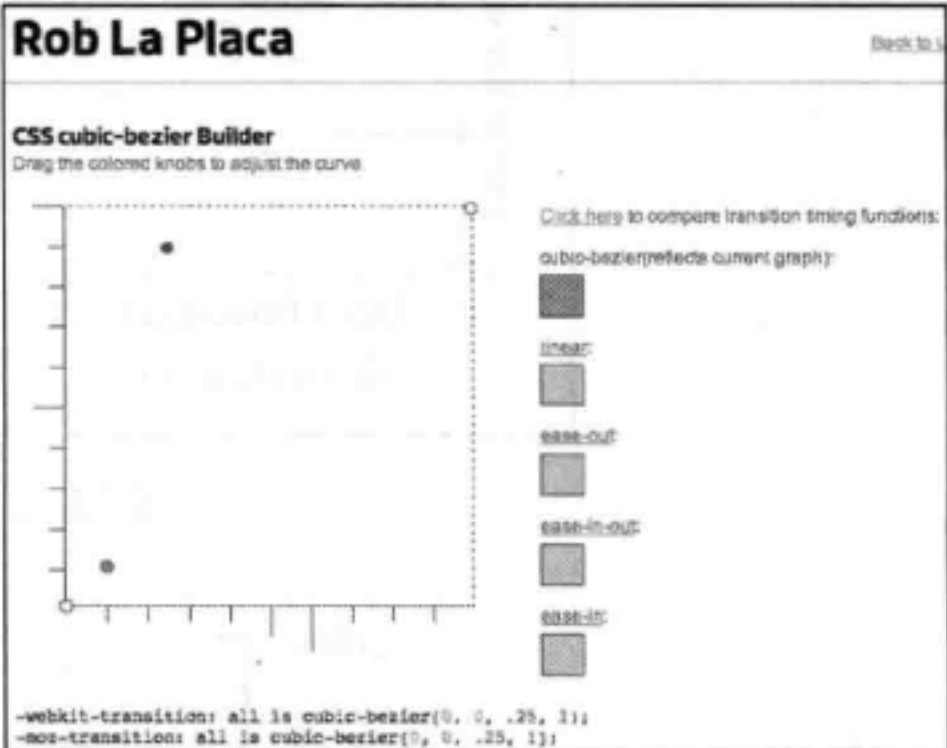
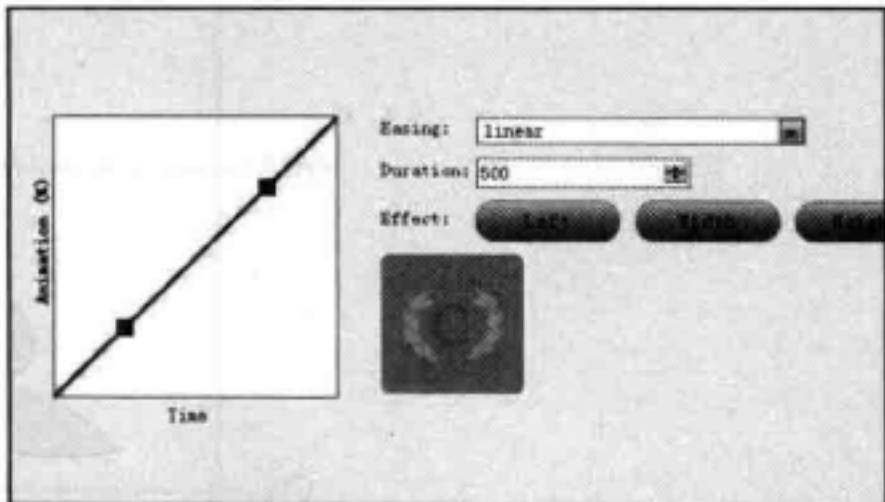


图 12-11 easing function 曲线

使用三次贝塞尔曲线函数，可以自定义动画的过渡速度。互联网上可以找到很多在线制作三次贝塞尔曲线的工具帮助我们绘制这些曲线，如表 12-4 所示[Ⓐ]。

[Ⓐ] 其他相关工具可以参阅 <http://www.w3cplus.com/source/front-end-developer-excellent-tool.html>。

表 12-4 在线制作三次贝塞尔曲线工具

工具名	图例
cubic-bezier (http://cubic-bezier.com/)	
cubic-bezier Builder (http://www.roblaplaca.com/examples/bezierBuilder/)	
Easing animation tool (http://matthewlein.com/ceaser/)	

3. step() 函数

step() 函数用于把整个操作领域划成同样大小的间隔，每个间隔都是相等的。该函数还指定发生在开始或结束的时间间隔是否另外输出百分比（换句话说，如果输出的百分比为 0% 表示输入变化的初始点）。看看来自 W3C 官网对 step() 函数的剖析图，如图 12-12 所示。

step() 函数非常独特，它允许在固定的间隔播放动画。例如，在 step() 函数图上可以看出，动画属性比在 0% 处开始，涨到 50%，动画结束时，属性值达到 100%（也就是结束状态属性）。并且在各个 step() 函数中每个步骤在过渡动画中不够平滑，如图 12-13 所示。

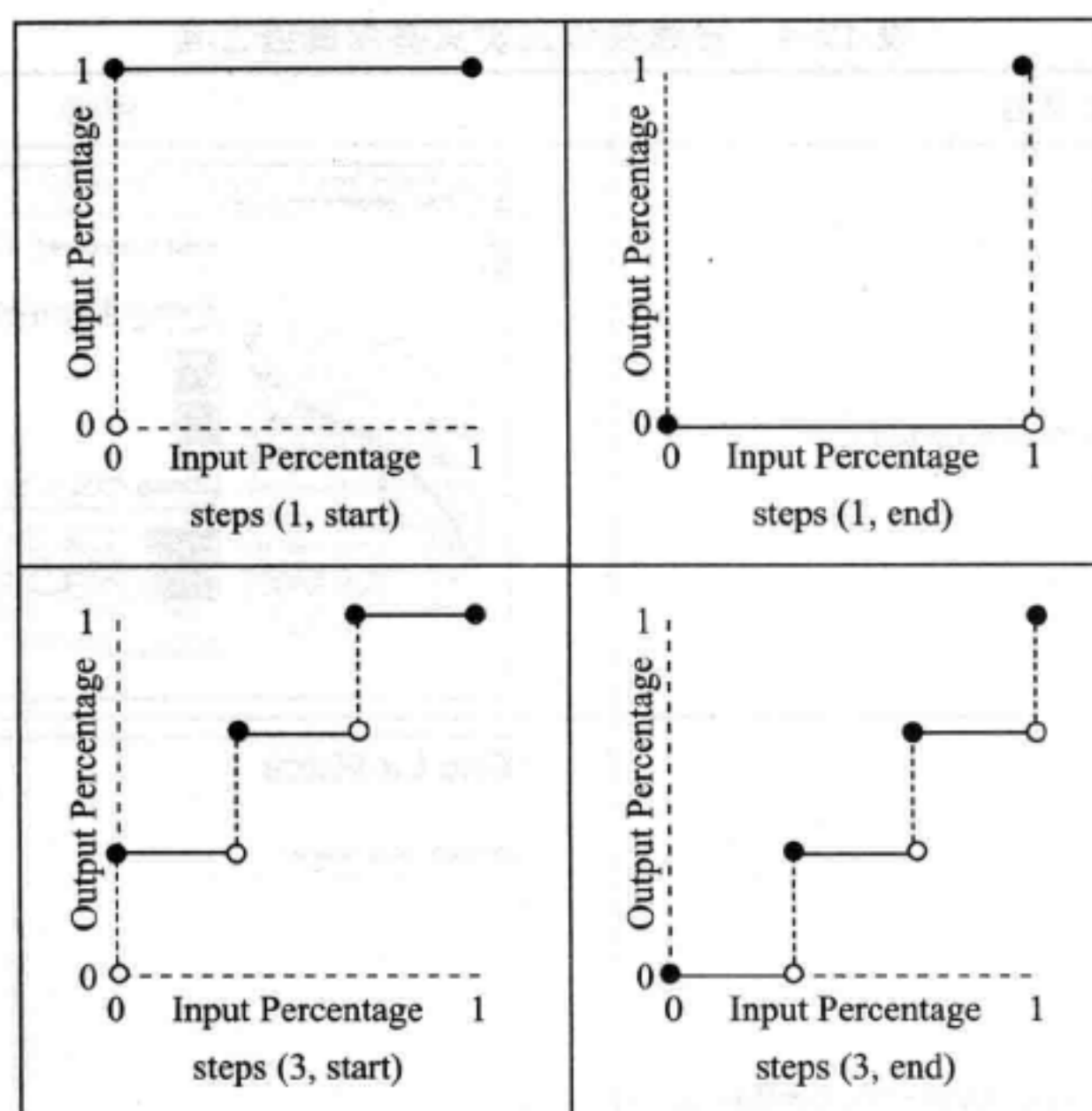


图 12-12 step() 函数

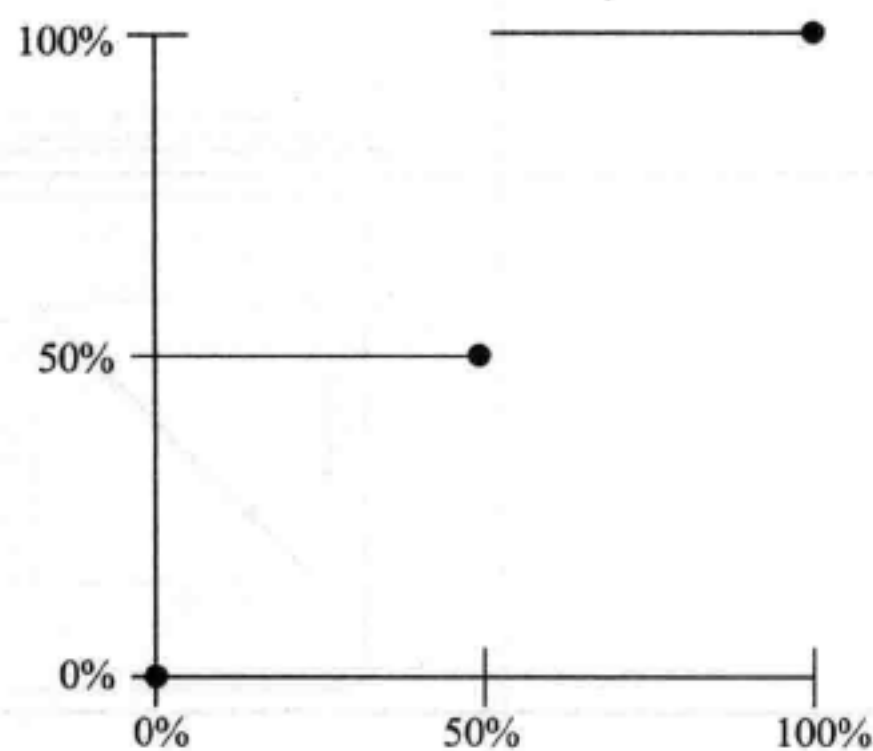


图 12-13 step() 函数

该函数语法如下。

```
step(<integer>[, [start|end]]?)
```

step() 函数主要包括两个参数。

- 第一个参数是一个数值 <integer>，主要用来指定 step() 函数间隔的数量，此值必须是一个大于 0 的正整数。
- 第二个参数是可选的，是 start 或 end，如果第二个参数忽略，则默认为 end 值。

其中 `step(1,start)` 相当于 `step-start`；`step(1,end)` 相当于 `step-end`。例如，如果想要动画有七个步骤，且动画结束的发生在最后一步，可以这样设置 `step()` 函数。

```
transition-timing-function: step(7,end)
```

`step(7,end)` 函数的动画过程如图 12-14 所示。

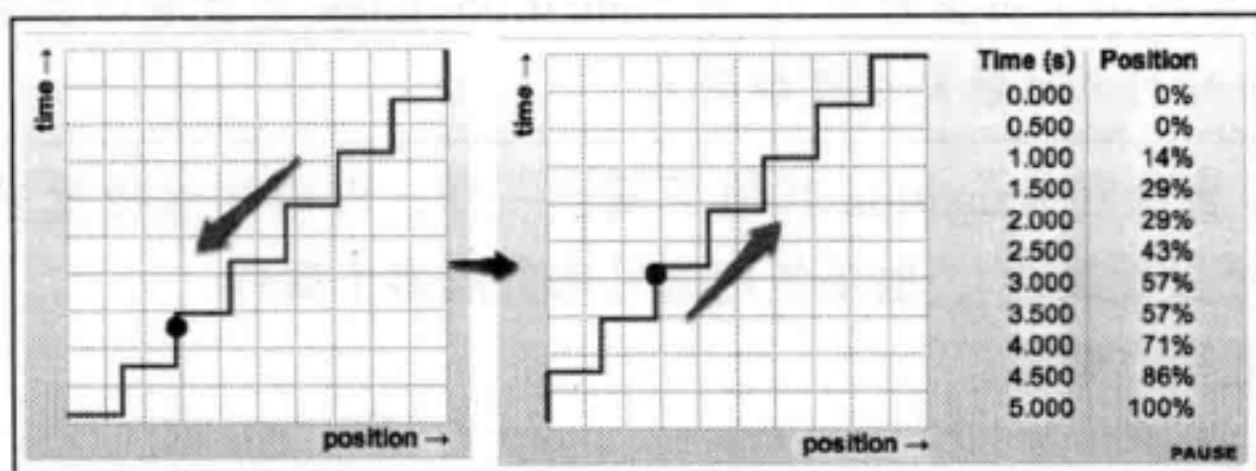


图 12-14 transition-timing-function 取值 `step(7,end)` 动画曲线

接下来看一个 `step(4,start)` 制作的动画效果，如图 12-15 所示。

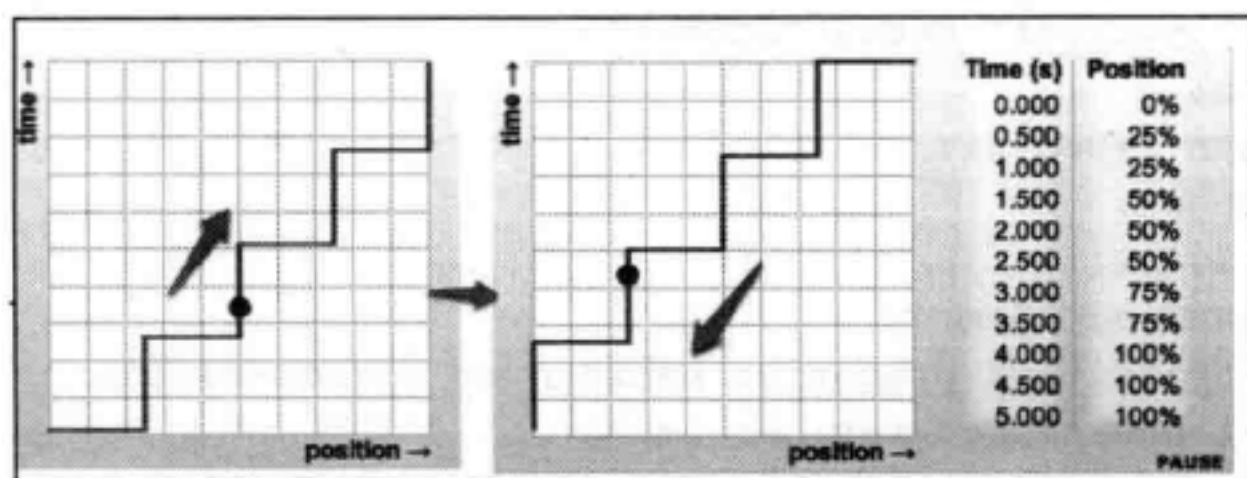


图 12-15 transition-timing-function:step(4,start)

当使用多个过渡属性 `transition-property` 时，也可以为每个过渡属性指定对应的过渡函数，当指定多个过渡函数时，需要用逗号将它们分隔开。如果有多个过渡属性，但只指定一个过渡函数时，这个过渡函数将应用于所有的过渡属性。

12.2.4 指定过渡延迟时间 transition-delay

前面介绍了三个属性，过渡属性 `transition` 还有一个过渡延迟时间属性 `transition-delay`，用来定义过渡延迟时间。该属性的基本语法如下。

```
transition-delay:<time>[,<time>]*
```

`transition-delay` 用来指定一个动画开始执行的时间，也就是说当改变元素属性值后多长时间开始执行过渡效果，其取值为 `<time>`，它可以是正整数、负整数和 0，非零的时候必须将单位设置为 s（秒）或者 ms（微秒）。

- 正整数：元素的过渡动作不会立即触发，当过了设定的时间值之后才触发。
- 负整数：元素的过渡动作会从该时间点开始显示，之前的动作被截断。

□ 0：元素的过渡动作会立即触发，没有任何延迟。在默认情况下，transition-delay 取值为 0。

 **提示**

`transition-duration` 和 `transition-delay` 在 `transition` 属性中都指的是时间，其使用方法也基本相似，不同的是 `transition-duration` 是过渡动画完成所需要的时间（也就是完成过渡动画总共用了多少时间）；而 `transition-delay` 是过渡动画在什么时间之后触发（也就是多少时间之后触发过渡动画）。

transition-delay 属性和 transition 其他子属性一样，可以同时设置多个属性（前提是为元素同时设置了多个过渡属性），也是使用逗号来分隔多个属性。

在上例的基础上稍作调整。

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>transition-delay 的基本应用 </title>
  <style type="text/css" media="screen">
    .transition{
      background: #8ec63f;
      width: 100px;
      height: 100px;
      border-radius: .5em;
      -webkit-transition-property: background,border-radius;
      -webkit-transition-duration: 2s,3s;
      -webkit-transition-timing-function: linear,ease-in;
      -webkit-transition-delay: 2s,4s;
      -moz-transition-property: background,border-radius;
      -moz-transition-duration: 2s,3s;
      -moz-transition-timing-function: linear,ease-in;
      -moz-transition-delay: 2s,4s;
      -o-transition-property: background,border-radius;
      -o-transition-duration: 2s,3s;
      -o-transition-timing-function: linear,ease-in;
      -o-transition-delay: 2s,4s;
      -ms-transition-property: background,border-radius;
      -ms-transition-duration: 2s,3s;
      -ms-transition-timing-function: linear,ease-in;
      -ms-transition-delay: 2s,4s;
      transition-property: background,border-radius;
      transition-duration: 2s,3s;
      transition-timing-function: linear,ease-in;
      transition-delay: 2s,4s;
    }
    .transition:hover {
      background:#f7941d;
      border-radius: 50%;
    }
  </style>
</head>
<body>
  <div class="transition">
  </div>
</body>
</html>
```

```

    </style>
</head>
<body>
    <div class="transition"></div>
</body>
</html>

```

在这个示例中同时给元素 `div.transition` 设置两个过渡属性 `background` 和 `border-radius`, 设置 `background` 属性过渡动画延迟 2 秒执行, 而 `border-radius` 属性过渡动画延迟 4 秒执行。整个过渡效果是, 当鼠标悬停在元素 `div.transition` 上时, 2 秒后触发元素的 `background` 属性从 “#8ec63f” 色过渡到 “#f7941d” 色, 在这个过渡中总共使用了 2 秒时间; 同时过了 4 秒后, 元素的 `border-radius` 属性从 “.5em” 过渡到 “50%”, 而这个过渡动画总共使用了 4 秒。简单点说, 鼠标悬浮在元素上 2 秒后, 会看到背景色从绿色逐渐过渡到橙色, 4 秒后, 会看到圆角矩形逐渐过渡到圆形。

12.2.5 多个 CSS3 过渡效果

很多时候, 不只想改变一个 CSS 的属性效果, 还想改变两个或者多个 CSS 属性效果。这在 `transition` 过渡中并不是很复杂的一件事情。可以使用上面讨论的 `transition-property` 普通书写语法, 用逗号分隔要过渡的各个 CSS 属性, 从而声明多个过渡。反过来, 其他普通书写值允许包含逗号分隔值, 其对应于 `transition-property` 值。

简写语法进一步简化此操作, 能够同时声明多项属性过渡, 只需要用逗号分隔声明。正如前面的示例所示。

```

.transition{
    transition: background 2s linear 2s, border-radius 3s ease-in 4s;
}

```

如果愿意, 可以在一行上显示多个属性, 或者为提高可读性使各个过渡属性各占一行, 如上面的示例所示。

使用多个过渡, 并且为 `transition-duration`、`transition-timing-function` 和 `transition-delay` 三个属性对的每个过渡属性 `transition-property` 设置相同值时, 在普通语法中, 只需要 `transition-property` 用逗号分隔每个过渡属性, 而其他的属性只设置一个就可以。要是简写语法, 这时就算每个过渡 CSS 属性都具有相同的 `transition-duration`、`transition-timing-function` 和 `transition-delay` 属性, 也必须显式设置, 如下面的代码所示。

```

/*transition 的普通写法*/
.transition {
    transition-property: width, background-color, height;
    transition-duration: 2s;
    transition-timing-function: ease;
    transition-delay: .5s;
}

```



```

}
/* 上面的代码效果等同于（简化写法）*/
.transition{
  transition: width 2s ease .5s, background-color 2s ease .5s,
             height 2s ease .5s;
}

```

12.3 CSS3 触发过渡

掌握了 CSS3 的过渡 transition 语法，在实际应用中应当如何触发过渡呢？上面所述的示例定义了过渡操作的各个层面，但却并没有提及触发过渡的时间和方法。因此，就目前的情况而言，单纯通过代码不会触发任何过渡操作。这一点和 JavaScript 中的过渡效果类似，需要通过用户的行为（如点击、悬浮）触发。

12.3.1 伪元素触发

使用 transition 时，常用鼠标悬浮（:hover）来触发过渡。加上“:hover”触发器后，代码如下所示。

```

.transition{
  background-color: green;
  width: 100px;
  height: 100px;
  transition: width 2s ease .5s,
             background-color 2s ease .5s,
             height 2s ease .5s;
}
.transition:hover {
  background-color: orange;
  width: 200px;
  height: 50px;
}

```

有了这个代码，当用户将鼠标悬浮在该元素上时，元素的宽度、背景色和高度会在经过 0.5 秒的初始延迟后，于 2 秒内动态从 100px 宽度变成 200px，100px 高变成 50px，背景色从绿色变成橙色。

事实上，触发过渡与触发器（在上述情况下，触发器为 :hover）本身没有太大关系。实际触发过渡的是元素状态变化。

在上面的示例中，状态变化是指宽度由窄变宽、由高变矮，同时绿色背景颜色变为橙色背景颜色。凑巧的是，状态变化作为“:hover”事件的结果发生。

为说明这与触发过渡的实际事件毫无关联，也为强调状态变化的重要性，下面提供了一些触发过渡的其他方法。

1. 使用 :active

“:active”伪类表示用户单击某个元素并按住鼠标按钮时显示的状态。以下示例中，当用户单击并按住元素时，发生宽度、高度和背景色属性过渡，因此该元素保持“活动”状态。

```
.transition{
    background-color: green;
    width: 100px;
    height: 100px;
    transition: width 2s ease .5s,
        background-color 2s ease .5s,
        height 2s ease .5s;
}
.transition:active {
    background-color: orange;
    width: 200px;
    height: 50px;
}
```

2. 使用 :focus

“:focus”伪类通常会在元素接收键盘焦点时出现。文本输入框元素上将发生过渡，且该元素得到焦点时会执行文本输入框元素宽度和背景色过渡。该代码如下所示。

```
input {
    background-color: green;
    width: 100px;
    transition: background-color 2s ease .5s,
        width 2s ease .5s;
}
input:focus {
    background-color: orange;
    width: 200px;
}
```

此处作为一个边点，当对“:hover”伪类应用过渡时，最好将“:focus”添加到选择器堆栈。这样，将能够丰富鼠标用户和键盘用户的体验。

3. 使用 :checked

“:checked”伪类在发生以下状况时触发过渡。文本框选中时，“span”元素宽度发生过渡。

```
input[type="radio"] ~ span {
    width: 100px;
    transition: width 2s ease-in 1s;
}
input[type="radio"]:checked ~ span {
    width: 200px;
}
```

12.3.2 媒体查询触发

触发元素状态变化的另一种方法是使用 CSS3 媒体查询 (Media Queries)。如果学习过媒体查询, 那么一定知道它能够根据某些元素 (比如设备宽度和方向) 更改应用于元素的样式。媒体查询过渡如下所示。

```
.transition{
    background-color: green;
    width: 100px;
    height: 100px;
    transition: width 2s ease .5s,
        background-color 2s ease .5s,
        height 2s ease .5s;
}
@media only screen and (max-width: 960px) {
    .transition{
        background-color: orange;
        width: 200px;
        height: 50px;
    }
}
```

在这个示例中, 元素呈现的是一个 100px 的绿色的正方形。但是, 如果用户将自己的窗口大小调整到 960px 或以下, 则元素将过渡为宽 200px、高 50px 的橙色长方形。当窗口超过 960px 的阈值后, 将会发生过渡。

当然, 如果网页加载时用户的窗口大小是 960px 或以下, 浏览器会在该部分应用这些样式, 但是由于不会出现状态变化, 因此不会发生过渡。

正如在上面的示例中所见, 基本上可以通过在某种程度上更改元素 CSS 的任何事件触发过渡。因此, 只要更改的属性是动画属性, 就会发生过渡。

12.3.3 JavaScript 触发

如果可以基于 CSS 的状态更改触发过渡, 自然可以通过 JavaScript 做到这一点。在下面的示例中, 使用纯粹的 CSS 示例时同样也会发生过渡。也就是说, 它是 CSS 状态变化的结果, 但是这次是通过 JavaScript 触发的过渡。

下面的示例使用简单的 jQuery 脚本切换元素类名称。

```
$(function(){
    $("#button").click(function(){
        $(".box").toggleClass("on");
    });
});
```

假设标记中具有包含 box 类的元素和 button 元素 ID, 则每次用户单击 “#button” 元素时, 此脚本都会将 on 类切换为 box。

尽管其本身没有任何作用，但可以向 on 类添加过渡，这样每次添加或删除类时，在两个声明中指定的值之间来回进行宽度和高度过渡，如下所示。

```
.box {
    width:200px;
    height: 200px;
    transition: width 2s linear 1s,height 2s ease 1s;
}
.box.on {
    width: 300px;
    height: 300px;
}
```

同样，此代码段重点强调样式发生导致过渡的点，可以通过任意数量的方法触发这些更改，包括 JavaScript。在考虑是否要使用 CSS 过渡替换 JavaScript 时需要牢记：事件通常应用通过 JavaScript 触发，简单动画或过渡则应使用 CSS 触发。当然，这只是一般性的指导原则，不一定是最佳选择，具体应用视条件而定。

12.4 CSS3 过渡技巧

现在，已经对过渡语法及其使用有了很好的掌握，接下来介绍 transition 中一些技巧，或许在将来的项目中可以派上用场。

12.4.1 一个完整的过渡

CSS3 过渡 transition 是异步运行的，该规范提供了 TransitionEnd 事件，允许 JavaScript 来同步一种过渡的结束属性。遗憾的是，该规范并没有很具体地阐述这个事件。事实上，它只是说明每个过渡属性的事件状态。如果用一个词来描述这种情况，你的“噩梦”就要开始了。

transition 规范中说 transition 运行简写属性（如 padding），会覆盖所有属性（padding-top、padding-right 等），但并没有说 TransitionEnd 事件是哪个属性。在 Gecko、Trident 和 Presto 内核浏览器中，即使过渡指定的是简写属性（例如 padding），但事件触发的普通写法的子属性（如 padding-top），可是在 WebKit 中会抓住机会将事情搞砸。如果指定 transition-property:padding，WebKit 将触发一个 padding 事件，但指定 transition-property:all 时，WebKit 将触发的事件为 padding-left 等。因为某些原因，当指定的过渡属性为 padding 时，在 iPhone 的 Safari 6.0.1 浏览器中除了触发 padding 属性之外还会触发 font-size 和 line-height 属性。

```
.example {
    padding: 1px;
    transition-property: padding;
```

```

    transition-duration: 1s;
}
.example:hover {
    padding: 10px;
}

```

上面的 CSS 在不同的浏览器中将触发不同的 TransitionEnd 事件。

❑ 在 Gecko、Trident 和 Presto 浏览器中触发 padding-top、padding-right、padding-bottom 和 padding-left 事件。

❑ 在 WebKit 浏览器中触发 padding 事件。

看另外一个例子。

```

.example {
    padding: 1px;
    transition-property: all, padding;
    transition-duration: 1s;
}
.example:hover {
    padding: 10px;
}

```

上面的 CSS 在不同的浏览器中将触发不同的 TransitionEnd 事件。

❑ 在 Gecko、Trident、Presto 和 WebKit 内核浏览器中将触发 padding-top、padding-right、padding-bottom 和 padding-left 属性。

❑ 在 iPhone 的 Safari 6.0.1 浏览器中将触发 padding-top、padding-right、padding-bottom、padding-left、font-size 和 line-height 属性。

介绍 transition 中的 transition-delay 属性时得知可以为这个属性设置负值，用来指定多长时间后触发过渡发生。例如，“transition-duration:1s; transition-delay:-1s”将会发生什么呢？在 WebKit 和 Gecko 内核浏览器中会立即触发过渡动作发生，而在 Trident 和 Presto 内核浏览器却不会转眼间触发过渡动作发生。

在 WebKit 和 IE 中对 background-position 属性开启了一个未知扩展，在触发 background-position 的 TransitionEnd 事件时，将指定的是 background-position-x 和 background-position-y，来替代 background-position 属性。

所以，即使知道过渡可以发生转变，但不能依赖于 TransitionEnd。因为它返回的只是一个只读字符串。虽然可以编写大量的 JavaScript 来实现这些效果，但不能这样做，因为没有恰当的功能来检测每一个属性，甚至可能包括一些不知道的属性。

12.4.2 可过渡的属性

规范中列出了很多能让浏览器支持动画过渡行为的 CSS 属性。正如前面介绍 transition-property 属性时所列的属性清单，详细参见表 12-2 所示。但这个列表包含的属性都是

CSS2.1 版本的 CSS 属性。任何更新的、可支持动画过渡行为的属性都没有列入规范中，如伸缩性盒模型中的 `order` 属性。

属性值的类型是一个很重要的因素。例如，`margin-top` 同时可以接受 `<length>` 和 `<percentage>` 的属性值，但根据可过渡的 CSS 属性列表，只有 `<length>` 值才具有动画效果。但是无论如何也没有办法让浏览器厂商使用 `transition` 支持取值为 `<percentage>` 的 CSS 属性具有动画效果。该规范包括了 `<percentage>` 的值，但在撰写本书的时候，还没有浏览器能够支持它的过渡动画。

12.4.3 优先的过渡属性

该规范在定义 `transition-property` 属性状态时，允许同时给一个属性定义多次。所以，可以定义 `padding` 的过渡为 1 秒，同时可以定义 `padding-left` 的过渡需要 2 秒；或者通过 `transition-property:all` 来定义一个默认的过渡风格，或者用其来覆盖一些特定的过渡属性。

在 Firefox 和 IE 浏览器中，这个工作做得很好。但在 Opera 浏览器下有一个优先秩序，而不是简单使用多个过渡属性列表中的最后一个适用的属性，其中 `padding-left` 属性优先于 `padding` 和 `all`。

真正麻烦的是在 WebKit 内核的浏览器下，如果一个属性设置了多次的过渡，那么不知道该怎么多次执行一个过渡。例如在 WebKit 下 `transition-duration:0.1s` 时段内执行“`transition-property:padding,padding-left;`”（警告，这不是一个好方法），WebKit 将呈现过渡至少两次。但真正的完美 `TransitionEnd` 事件，其中可以设置数以百计的过渡。

12.4.4 过渡的开始和结束为 auto

在 CSS 中，有些属性可以取值为 `auto`。当属性为 `auto` 时，浏览器会自动计算出一个合理的值。例如段落“`<p>`”或者任何块级元素，如果设置了 `width:auto` 时，它们的宽度将等于它们的父元素宽度。有时候可能会通过 `transition` 属性将 `width:auto` 值过渡到一个指定的宽度值。在可过渡性属性的规范中，`auto` 值既不会执行也不会说不执行。

在 Firefox、IE 和 Opera 浏览器下过渡不会执行属性值为 `auto` 的属性。另一方面，在 WebKit 内核浏览器下几乎所有具有属性值为 `auto` 的属性都可以过渡。

对于其他属性，比如宽度和高度，在 WebKit 内核的浏览器下并不完全如您所愿。如果 `width:auto` 计算出来的宽度为 300px，并且过渡到 100px，这个时候过渡并不是从 300px 过渡到 100px。

12.4.5 隐式过渡

“隐式过渡”指的是，当一个属性改变时引起另一个属性到一个属性的过渡。很困惑吧！来看一个简单的例子，帮助我们解决这个困惑。考虑“`font-size: 18px;`”

padding:2em;”——padding 值通过“2 * font-size”得到。这就是 em 的值，给的 padding 为 36px。

相对值类型包括：<percentage>、<length>、em、rem、vh、vw 等。使用一个相对值，例如 padding:2em，使用浏览器每次根据不同的值（如 font-size）来计算属性的 getComputedValue() 值。这反过来为 padding 触发了一个过渡，因为计算的方式改变了。这种过渡称为“隐式过渡”，因为 padding 属性并没有被明确修改。

大多数浏览器都支持“隐式过渡”，唯一例外的是 IE10，它仅支持 line-height 的“隐式过渡”。在 WebKit 内核浏览器下，“隐式过渡”可以运用于除 vertical-align 属性之外的所有属性。除了字体相对属性值外，还有宽度相对属性值（通常可见的是 <percentage>）、视窗相对属性值（如 vh 和 vw）、默认的初始值（如 Opera 下的 column-gap:1em）和当前颜色。所有这些都有可能触发“隐式过渡”。

12.4.6 开关状态的不同过渡方式

在下面的代码示例中，开关状态均会导致背景颜色的过渡。

```
.example {  
  background-color: blue;  
  transition: background-color 2s linear 1s;  
}  
.example:hover {  
  background-color: green;  
}
```

更改关闭状态的持续时间和计时函数，如下所示。

```
.example {  
  background-color: blue;  
  transition: background-color 1s ease-out;  
}  
.example:hover {  
  background-color: green;  
  transition: background-color 2s linear;  
}
```

在这段代码中，将最初的声明转为“:hover”状态，并在触发“:hover”前向元素应用的样式添加了一行新代码。现在，“开启”过渡需要 2 秒（指定为 2s linear），“关闭”过渡需要 1 秒（指定为 1s ease-out）。

这个想法的关键在于，指定浏览器当元素从绿色过渡到蓝色时，使用 1s ease-out，但当元素从蓝色过渡到绿色时，则使用 2s linear。

这有助于提高各种 UI 元素的可用性。例如，如果过渡时间超过 4 秒（有点冗长），可以在更短时间内使用元素返回正常的“关闭”状态。

12.4.7 几乎无限延迟的过渡

另一项关于过渡的技巧是使用无限延迟模拟永久状态更改，并且无须使用 JavaScript 脚本代码，如下所示。

```
.example {
  width: 100px;
  height: 100px;
  background-color: blue;
  transition: width 0s ease-out 999999s, height 0s ease-out 999999s;
}
.example:active {
  width: 200px;
  height: 200px;
  background-color: green;
  transition: width 2s, height 2s;
}
```

在这个示例中，使用“:active”更改元素的宽度和高度。开启状态的持续时间为 2 秒，然而，该元素过渡回到原始尺寸（100px × 100px）出现的延迟持续时间为 999999 秒和 0 秒。但是，其持续时间无关紧要，因为基本上网页时间延迟无上限，相当于 12 天。

这就意味着关闭状态（或者触发“:active”状态前的原始元素状态）永远不会出现，除非用户待机长达 12 天。因此，结果是“:active”变为永久性状态，从而使人产生 JavaScript 改变元素样式的错觉。

12.4.8 通过硬件加速过渡更加流畅

有时，基于 CSS 的过渡动画并没有所期待的那样流畅。有时，动画元素会在播放动画后留下“痕迹”。这通常是一个 WebKit 问题，在基于 iOS 的设备上尤其明显。可以通过支持硬件加速来提高 WebKit 的过渡性能。下面是采用“-webkit-”前缀实现简单过渡的代码。

```
.example {
  background-color: blue;
  width: 100px;
  height: 200px;
  -webkit-transition: width 2s ease,
    height 2s ease;
}
.example:hover {
  height: 300px;
  width: 300px;
}
```

这个示例将一个宽高为 200 像素的蓝色正方形过渡到一个宽高为 300 像素的正方形。在 Chrome 浏览器中，这将会导致滞后痕迹，如图 12-16 所示。

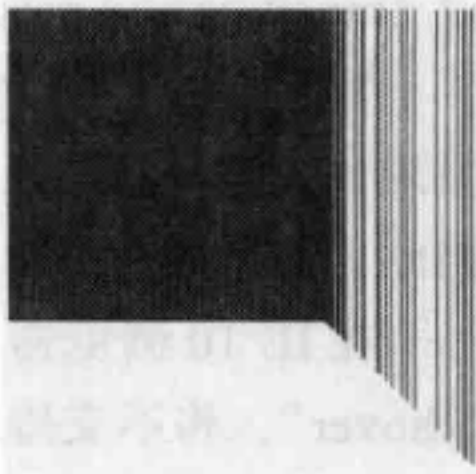


图 12-16 过渡期间 Chrome 中的滞后痕迹

为解决此问题，必须通过添加一行 CSS 代码来启用硬件加速。下面显示了更正后的代码。

```
.example {
  background-color: blue;
  width: 100px;
  height: 200px;
  -webkit-transition: width 2s ease,
    height 2s ease;
  -webkit-transform: translate(0);
}
.example:hover {
  height: 300px;
  width: 300px;
}
```

给该过渡元素添加了一个 transform 属性，并将其值设置为 translateZ(0)。该函数本身通常用于“变形”或定位 Z 轴（在 3D 变形空间中）上的元素，但 translateZ() 取值为 0 时妨碍任何实际的新定位。此函数带有硬件加速的元素过渡，能够简化过渡并消除痕迹或条纹，但 0 值不会重新定位该元素。

12.4.9 过渡和伪元素

第 2 章介绍了 CSS3 中的伪元素 (:after 和 :before)。伪元素 “:before” 和 “:after” 可以在页面中生成一些内容，如果还不太熟悉如何通过 CSS 样式来生成内容，建议阅读第 2 章的伪元素和生成内容。

虽然 CSS3 还定义了额外的伪元素 (:alternate、:outside)，但它们到目前并未得到浏览器厂商的友好支持。按理说，所有具有动画的功能的 CSS 属性也应该可以运用于伪元素的过渡动画，但现实与理想并不是完全吻合的，在一些浏览器中对伪元素的过渡动画功能并不完全支持。

在 Firefox 和 IE 10 浏览器中，伪元素的过渡动画得到很好的支持，但在 Opera、Chrome 和 Safari 浏览器中，伪元素的过渡动画支持度并不很好。伪元素生成内容的过渡动画自身就带有一些问题。TransitionEnd 事件并不兼顾它们。或许在将来某个时候，可以通过 TransitionEnd.pseudoElement 给伪元素提供触发过渡动画。

曾经有一段时间，通过改变 content 的值来生成内容，在某些情况下在 IE 8 中会重新呈现该元素（如进入一个 :hover 状态）。原来，修复旧的 IE 效果，却干扰了所有的浏览器。因此，当试图在一个伪元素上生成的内容上使用过渡属性，要确保内容没有改变。

在 IE 10 浏览器中，如果主体元素没有 “:hover” 状态，在伪元素生成的内容中使用 “:hover”，将不支持过渡动画效果。

```
.example:before {
  content: "hello";
  color: red;
```



```

    transition: all 1s linear 0s;
}
.example:hover:before {
    color: green;
}
/* 下面的规则对于 IE10 的伪元素在 :hover 状态使用过渡是必须要的 */
.example:hover {
}

```

奇怪的是，在主元素上“:hover”并不是必需的（可能为空）。如果没有设置这个样式规则，在 IE 10 下会将“:hover”解析为“:active”状态效果（也就是用户鼠标点击元素那一刻的效果）。更令人惊奇的是这个“:active”状态会一直持续下去，甚至在删除之后也依旧存在，要想彻底删除它只能由其他点击事件来覆盖。

12.5 综合案例：纯 CSS3 制作 CSS Dock 导航效果

使用 Mac 机人对其导航效果肯定很熟悉，这种导航效果称为 CSS Dock 导航效果。早前在 Web 页面中就实现了这种导航效果，但其必须依赖于 JavaScript 脚本。自从 CSS3 的出现后，可以不在依赖 JavaScript 脚本的情况下实现这种优雅的效果。

在下面的这个实战的案例中，将借助 CSS3 的一些新特性来模拟 iOS 系统桌面的 Dock 导航效果，如图 12-17 所示。

在这个效果中，不仅使用了 CSS3 的 transition 属性，还使用了很多前面介绍的属性，例如选择器中的伪类（:before 和 :after）实现 tip 效果；变形 transform 制作面板；阴影和圆角属性；另外还使用了将要介绍的 CSS3 动画属性 animation 以及



☆图 12-17 纯 CSS 模拟 iOS 系统
桌面 Dock 导航

及 WebKit 独有的镜像属性 box-reflec 等。回到本节的主题中来，通过实例介绍 CSS3 的 transition 属性在实际实现的效果。

实现 CSS Dock 导航的 HTML 结构非常简单，采用的是一个列表，每个列表项放置一个导航项目，并且内置了导航项的提示信息。而导航项目使用的是常见的 3D 空间布局结构，具体代码如下所示。

```

<div class="wrapper">
  <div class="content">
    <h1>CSS Dock</h1>
    <!-- CSS Dock 导航项 -->
    <div class="dock">
      <ul>
        <li id="mail">

```

```

    <a href="#mail">
      <em><span>Mail</span></em>
      
    </a>
  </li>
  <li id="ical">
    <a href="#ical">
      <em><span>iCal</span></em>
      
    </a>
  </li>
  <li id="addressbook">
    <a href="#addressbook">
      <em><span>Address Book</span></em>
      
    </a>
  </li>
  <li id="iphoto">
    <a href="#iphoto">
      <em><span>iPhoto</span></em>
      
    </a>
  </li>
  <li id="idisk">
    <a href="#idisk">
      <em><span>iDisk</span></em>
      
    </a>
  </li>
</ul>
</div>
<!-- CSS Dock 导航项底部面板 -->
<div class="dock-stage">
  <div class="dock-container">
    <div class="dock-panel"></div>
  </div>
</div>
</div>

```

在开始写 CSS Dock 导航前，先完成一些基本样式，方便后面效果的完善。

```

body {
  margin: 0;
  padding: 0;
  font: normal 14px/1.5 "PT Sans", Helvetica, Arial, sans-serif;
  background: rgb(30, 30, 30);
  color: rgb(220, 220, 220);
}

.wrapper {
  position: absolute;

```

```

    top: 48%;
    left: 50%;
    width: 660px;
    height: 340px;
    margin: -190px 0 0 -330px;
}

.content {
    position: relative;
    width: 660px;
    height: 340px;
    overflow: hidden;
    background: rgb(15, 15, 15) url(images/bg.jpg) no-repeat center bottom;
    border-radius: 10px;
    box-shadow: 0 1px 10px rgb(10, 10, 10);
}

h1 {
    margin-top: 1.5em;
    color: rgba(255, 255, 255, .9);
    line-height: 1;
    font-size: 70px;
    text-align: center;
    font-weight: bold;
    text-shadow: 0px 1px 5px rgb(255, 160, 55), 0px 1px 15px rgb(255, 160, 55);
    cursor: default;
}

h1 em {
    display: inline-block;
    font-size: .29em;
    font-style: normal;
    vertical-align: top;
    margin-left: -.4em;
    padding-top: .46em;
}

```

下面开始 CSS Dock 导航效果部分的实现，将要完成的效果主要有：

- ❑ 导航项目水平排列居中显示，并且具有翻转的倒影效果。
- ❑ 导航项被点击后会有有一个上下弹跳，并且底部有一个高亮圆点显示，表示当前导航项已被点击。
- ❑ 用户鼠标悬浮在导航项上时，会有一个 tips 的提示效果，同时导航项图片具有放大效果。
- ❑ 让所有导航项放置在一个面板上。

CSS Dock 导航的效果明确了，思路也有了，接下来就好办多了，首先使用 `display: inline-block` 让导航项水平排列，同时控制文本的对齐方式让导航项居中显示。再配合 `box-reflect` 属性实现导航项翻转的倒影效果，只不过到写这个案例时，仅有 WebKit 内核的浏览器支持“`box-reflect`”属性，也就是说在这个效果中，仅有 WebKit 内核浏览器才具有倒影效果。


```

.dock {
  position: absolute;
  bottom: 0;
  z-index: 10;
  width: 100%;
  text-align: center; /* 让导航项水平居中 */
  font: normal 14px/1 'Lucida Grande', Arial, sans-serif;
}

.dock ul {
  position: relative;
  display: inline-block;
  padding: 0 5px;
  margin: 0;
}

.dock li {
  display: inline-block; /* 让导航项水平显示 */
  position: relative;
  margin: 0 1px;
  margin-bottom: 15px;
  vertical-align: baseline; /* 导航项以基线对齐 */
  /* 在 WebKit 下实现导航项翻转倒影效果 */
  -webkit-box-reflect: below -16px -webkit-gradient(
    linear, left top, left bottom,
    from(transparent),
    color-stop(91%, rgba(255, 255, 255, .1)),
    color-stop(91.01%, transparent),
    to(transparent)
  );
}

.dock a {
  display: inline-block;
  cursor: default;
  outline: none;
}

```

第二个效果，使用目标选择器“:target”配合 CSS3 的 animation 属性，实现“用户点击导航项时，导航项会上下抖动”，并且通过伪类“:after”在单击的导航项添加一个圆点，这个圆点出现的过程使用了本章介绍的过渡属性 transition，让这个圆点用 0.5 秒时间从隐藏过渡到可见。

```

/* 使用 keyframes 自定义一个 bounce 动画 */
@-webkit-keyframes bounce {
  0% { -webkit-transform: translateY(0); }
  100% { -webkit-transform: translateY(-20px); }
}

@-moz-keyframes bounce {
  0% { -moz-transform: translateY(0); }
  100% { -moz-transform: translateY(-20px); }
}

```

```

}
@-o-keyframes bounce {
    0% { -o-transform: translateY(0); }
    100% { -o-transform: translateY(-20px); }
}

@-ms-keyframes bounce {
    0% { -ms-transform: translateY(0); }
    100% { -ms-transform: translateY(-20px); }
}

@keyframes bounce {
    0% { transform: translateY(0); }
    100% { transform: translateY(-20px); }
}

/* 用户点击导航项时触发 bounce 动画 */
.dock li:target a {
    -webkit-animation: bounce .3s 6 alternate ease-out;
    -moz-animation: bounce .3s 6 alternate ease-out;
    -o-animation: bounce .3s 6 alternate ease-out;
    -ms-animation: bounce .3s 6 alternate ease-out;
    animation: bounce .3s 6 alternate ease-out;
}

/* 通过伪类 ":after" 给点击的导航项底部添加一个圆点，以示被选择 */
.dock li:after {
    content: " ";
    position: absolute;
    bottom: -5px;
    left: 50%;
    width: 5px;
    height: 5px;
    opacity: 0;
    visibility: hidden;
    background-color: rgba(255, 255, 255, .8);
    margin-left: -2px;
    border-radius: 5px;
    box-shadow:
        inset 0 1px 3px rgba(75, 255, 255, .4),
        0 0 4px rgba(75, 255, 255, .5),
        0 -1px 7px rgb(75, 255, 255);
    /* 通过 transition 属性设置圆点花 .5s 时间从不可见过渡到可见状态 */
    -webkit-transition: opacity .5s;
    -moz-transition: opacity .5s;
    -o-transition: opacity .5s;
    -ms-transition: opacity .5s;
    transition: opacity .5s;
}

```

在 iOS 系统下的 Dock 导航效果中，当用户鼠标悬浮在导航项上时，导航项的应用图标会放大，并且会有一个揭示信息显示出来。

```

/*tooltips 效果*/
.dock em {

```

```

position: absolute;
top: -34px;
left: 50%;
display: none;
width: 150px;
margin-left: -75px;
text-align: center;
}
.dock em:after {
content: " ";
position: absolute;
top: 100%;
left: 50%;
margin-left: -6px;
width: 0;
height: 0;
border-left: 6px solid transparent;
border-right: 6px solid transparent;
border-top: 6px solid rgba(0, 0, 0, .6);
border-bottom: none;
}
.dock em span {
display: inline-block;
padding: 5px 12px;
font-size: 14px;
font-style: normal;
color: #FFF;
background: #000;
background: rgba(0, 0, 0, .6);
text-shadow: 1px 1px 1px rgba(0, 0, 0, .9);
border-radius: 12px;
}
.dock li:hover em,
.dock li a:focus em {
display: block;
}
.dock img {
width: 86px;
height: auto;
border: none;
/* 鼠标悬浮时修改图片的宽度和高度, 模拟一个图片放大效果 */
-webkit-transition: width .2s, height .2s;
-moz-transition: width .2s, height .2s;
-o-transition: width .2s, height .2s;
-ms-transition: width .2s, height .2s;
transition: width .2s, height .2s;
}
/* 鼠标悬浮时, 导航项图片放大 */
.dock li:hover img,
.dock li a:focus img {
width: 96px;
}
.dock li:active img {
opacity: .9;
}

```


这里是通过修改 icon 图标大小来实现图片放大假象。其实还可以通过 CSS3 的变形属性 transform 中的缩放 scale() 来实现图片放大功能。这样 CSS Dock 的导航效果就初步形成, 为了让效果更贴近 iOS 系统的 Dock 导航效果, 使用 div 标签配合 CSS3 3D transform 部分属性重新模拟制作了一个导航项底部效果。

```
.dock-container {
    -webkit-transform-style: preserve-3d;
    -moz-transform-style: preserve-3d;
    -ms-transform-style: preserve-3d;
    -o-transform-style: preserve-3d;
    transform-style: preserve-3d;
}
.dock-panel {
    width: 560px;
    height: 100px;
    background: -webkit-linear-gradient(to bottom, #eee 0%, #eee 28%, #bbb 51%, #bbb
100%);
    background: linear-gradient(to bottom, #eee 0%, #eee 28%, #bbb 51%, #bbb 100%);
    -webkit-backface-visibility: visible;
    -moz-backface-visibility: visible;
    -o-backface-visibility: visible;
    -ms-backface-visibility: visible;
    backface-visibility: visible;
    -webkit-transform-origin: 50% 50%;
    -moz-transform-origin: 50% 50%;
    -o-transform-origin: 50% 50%;
    -ms-transform-origin: 50% 50%;
    transform-origin: 50% 50%;
    -webkit-transform: perspective(800px) rotate3d(1, 0, 0, 50deg);
    -moz-transform: perspective(800px) rotate3d(1, 0, 0, 50deg);
    -o-transform: perspective(800px) rotate3d(1, 0, 0, 50deg);
    -ms-transform: perspective(800px) rotate3d(1, 0, 0, 50deg);
    position: relative;
    left: 50%;
    margin-left: -280px;
    bottom: -50px;
}
```

完成这一步之后, 就可以使用浏览器浏览到纯 CSS 实现 iOS 的 Dock 效果了。

12.6 本章小结

在这一章中主要介绍了 CSS3 的 transition 及其四个子属性 transition-property、transition-duration、transition-timing-function 和 transition-delay 的语法以及基本使用方法。最终结合 transition 在实际中的技巧与技术, 让大家深入了解如何使用 transition 属性制作过渡动画效果。

CSS3 除了 transition 属性可以模拟一些简单的动画交互效果之外, 还有一个属性 animation, 这个属性让你使用 CSS 实现动画效果更轻松, 接下来的一章中, 将带领大家深入了解 CSS3 的 animation 属性。

CSS3 动画

上一章详细介绍了使用 CSS3 的 `transition` 属性实现一些属性过渡的动画效果。虽然 `transition` 在一定的时间内可以实现元素的初始状态在指定的时间范围过渡最终状态，模拟一种过渡动画效果，但它的功能是非常有限的。因此，CSS3 新增了一个动画属性 `animation`。与过渡属性 `transition` 属性不同的是，CSS3 的 `animation` 属性可以像 Flash 制作动画一样，通过关键帧控制动画的每一步，实现更为复杂的动画效果。

13.1 CSS3 动画简介






如果以前接触过 Flash 动画，那么 CSS3 能立即上手；如果从未接触过 Flash 动画，只要稍稍学习一下也能很快掌握 CSS3 的动画。CSS3 中通过 `animation` 实现动画和 `transition` 实现动画非常类似，都是通过改变元素的属性值来实现动画效果的。它们的区别主要在于：使用 `transition` 属性只能通过指定属性的初始状态和结束状态，然后在两个状态之间进行平滑过渡的方式来实现动画。而 `animation` 实现动画效果主要由两个部分组成。

- 1) 通过类似 Flash 动画中的关键帧来声明一个动画；
- 2) 在 `animation` 属性中调用关键帧声明的动画，从而实现一个更为复杂的动画效果。

13.1.1 浏览器兼容性

`animation` 属性到目前为止得到了众多现代浏览器的支持，不过和很多 CSS3 属性一样，需要添加浏览器的私有前缀。其兼容性如表 13-1 所示。

表 13-1 animation 的浏览器兼容性

属性名称					
animation	10+ ✓	5.0+ ✓	4.0+ ✓	12+ ✓	4.0+ ✓

在现代浏览器以及移动端浏览器中，对 animation 属性的支持各有不同。

- ❑ 在 Chrome 4+、Safari 4.0+、Firefox 5.0+ 至 Firefox 16 浏览器中需要添加各浏览器的私有属性。
- ❑ 在 IE 10+、Firefox 16+ 和 Opera 12+ 浏览器中支持 animation 的标准语法。
- ❑ Opera16 +、IOS Safari 3.2+、Android Browser 2.1+、Blackberry Browser 7.0+ 浏览器使用 animation 需要添加前缀 -webkit。

13.1.2 CSS3 动画属性

在 CSS3 中可以通过 animation 创建复杂的动画序列，像 transition 属性一样用来控制 CSS 的属性实现动画效果。与 transition 属性相比，animation 属性可以通过 @keyframes 构建一些 transition 的动画效果，类似于 Flash 的动画效果。虽然目前 animation 属性还不能实现类似 Flash 制作的动画效果，相信未来不久，可以使用 CSS3 的 animation 属性设计出轻量级的 Flash 动画效果来。

第 12 章主要介绍了 transition 属性制作一些简单的属性过渡动画效果，但本章讨论的不是 transition 属性实现的元素属性过渡效果，这里主要讨论的是使用 @keyframes 实现的动画效果。

CSS3 动画属性 animation 和 CSS3 的 transition 属性一样是一个复合属性，它包含了众多子属性，如 animation-name、animation-duration、animation-timing-function、animation-delay、animation-iteration-count、animation-direction、animation-play-state 和 animation-fill-mode。其使用语法如下所示。

```
animation:[<animation-name> || <animation-duration> || <animation-timing-function> ||
<animation-delay> || <animation-iteration-count> || <animation-direction> ||
<animation-play-state> || <animation-fill-mode>] *
```

到目前为止，CSS3 共具有八个子属性，而每一个子属性都有其具体的含义与功能作用。接下来一起来看看各个子属性所起作用。

- ❑ animation-name：主要用来指定一个关键帧动画的名字，这个动画名必须对应一个 @keyframes 规则。CSS 加载时会应用 animation-name 指定的动画，从而执行动画。
- ❑ animation-duration：主要用来设置动画播放所需时间，一般以秒为单位。
- ❑ animation-timing-function：主要用来设置动画的播放方式，与 transition-timing-function 类似。
- ❑ animation-delay：主要用来指定动画开始时间，一般以秒为单位。

- ❑ `animation-iteration-count`: 主要用来指定动画播放的循环次数。
- ❑ `animation-direction`: 主要用来指定动画的播放方向。
- ❑ `animation-play-state`: 主要用来控制动画的播放状态。
- ❑ `animation-fill-mode`: 主要用来设置动画的时间外属性。

CSS3 动画属性具有 8 个子属性，很多情况之下都是分开来写，如下所示。

```
animation-name: moveten;
animation-duration: 1s;
animation-timing-function: steps(10,end);
animation-iteration-count: infinite;
animation-direction: alternate;
animation-delay: 3s;
animation-fill-mode: backwards;
```

如果加上各大浏览器兼容的私有属性前缀，这么一来就变得极其烦琐。幸运的是，所有的子属性都可以合成在一起，并且每个子属性之间使用空格隔开。

```
animation:<animation-name> <animation-duration> <animation-timing-function>
<animation-iteration-count> <animation-direction> <animation-delay>
<animation-fill-mode>
```

这样一来，上面示例可以简写成这样：

```
animation: moveten 1s steps(10,end) infinite alternate 3s backwards;
```

在 CSS3 动画应用中，除了可以将动画子属性简写在一起之外，还可以将多个动画应用在一个元素之上。同时将多个动画属性运用到一个元素之上时，可以包括每个动画名称的分组，每个简写的分组以逗号分隔开。例如：

```
animation: animation1 1s ease .5s backwards, animation2 2s ease 1s forwards;
```

13.2 关键帧

上面提到过，CSS3 的 `animation` 制作动画效果主要包括两部分，首先是用关键帧声明一个动画，其次是在 `animation` 调用关键帧声明的动画。所以先来了解第一步，使用关键帧声明动画。

在 CSS3 中，把 `@keyframes` 称为关键帧，玩过 Flash 的朋友可能对这个东西并不陌生。接下来介绍如何使用关键帧声明一个动画。

13.2.1 @keyframes 的作用

`transition` 制作一个简单的动画效果时，包括了元素的初始属性和最终属性，一个开始执行动作时间和一个延迟动作时间以及一个动作变换速率，其实这些值都是一个中间值，如果要控制得更细一些，比如说要第一个时间段执行什么动作，第二个时间段执行什么动

作(换到 Flash 制作动画中来说,就是第一帧要执行什么动作,第二帧执行什么动作),这样用 transition 就很难实现了,此时也需要一个“关键帧”来控制。在 CSS3 中就是通过 @keyframes 属性来实现这样的效果的。

13.2.2 @keyframes 的语法

@keyframes 具有其自己的语法规则,命名是由 @keyframes 开头,后面紧跟着是“动画的名称”加上一对花括号“{...}”,括号中就是不同时间段样式规则,有点像 CSS 的样式写法。一个 @keyframes 中的样式规则是由多个百分比构成的,如 0% ~ 100%,可以在这个规则中创建更多个百分比,分别给每个百分比中需要有动画效果的元素加上不同的属性,从而让元素达到一种不断变化的效果,比如说移动,再比如改变元素颜色、位置、大小和形状等。不过有一点需要注意,可以使用“from”“to”代表一个动画是从哪开始,到哪结束,也就是说 from 就相当于 0%,而 to 相当于 100%。值得一说的是,0% 不能像别的属性取值一样把百分比符号省略,在这里必须加上百分符号(%)。如果没有加上,这个 @keyframes 是无效的,不起任何作用。因为 @keyframes 的单位只接受百分比值。

@keyframes 可以指定任何顺序排列来决定 animation 动画变化的关键位置,具体语法规则如下。

```
keyframes-rule: '@keyframes' IDENT '{' keyframes-blocks '}';
keyframes-blocks: [keyframe-selectors block]*;
keyframe-selectors: ['from' | 'to' PERCENTAGE][', ' ['from' | 'to' | PERCENTAGE]]*;
```

把上面的语法综合起来。

```
@keyframes IDENT {
  from {
    /*CSS 样式写在这里 */
  }
  percentage {
    /*CSS 样式写在这里 */
  }
  to {
    /*CSS 样式写在这里 */
  }
}
```

也可以将关键词 from 和 to 换成百分比。

```
@keyframe IDENT {
  0% {
    /*CSS 样式写在这里 */
  }
  percentage {
    /*CSS 样式写在这里 */
  }
  100% {
```

```

    /*CSS 样式写在这里 */
}
}






```

其中 IDENT 就是一个动画名称，可以取一个任意定义的动画名称，当然语义化一点会更好。percentage 是一个百分比值，用来定义某个时间段的动画效果。

13.2.3 浏览器兼容性

@keyframes 在 animation 属性中是必不可少的一个属性，浏览器对 @keyframes 的兼容性直接影响到 animation 属性能在哪些浏览器下运行。在写本节内容之时，@keyframe 在众多现代浏览器中得到了较好的支持，除了部分浏览器中需要添加前缀“-webkit-”之外，其他浏览器都支持了 animation 属性的标准写法，详细的兼容性如表 13-2 所示。

表 13-2 @keyframes 浏览器兼容性

属性名称					
@keyframes	10+ ✓	5.0+ ✓	4.0+ ✓	4.0+ ✓	12.0+ ✓

针对早期的浏览器版本，要正常地让 @keyframes 起作用，同样需要添加不同浏览器的前缀。

- ❑ Chrome 4+、Safari 4+ 浏览器中需要添加前缀“-webkit-”；
- ❑ Firefox 5.0 ~ 21.0 版本中需要添加前缀“-moz-”；
- ❑ Opera 12.0 ~ 15.0 版本中需要添加前缀“-o-”，Opera15.0+ 版本需要添加前缀“-webkit”；
- ❑ IE 10+，Firefox 21+ 版本支持 @keyframes 的标准语法，不需要添加浏览器私有前缀；
- ❑ iOS 3.2、Android Browser 4.0+ 和 Blackberry Browser 7.0+ 版本需要添加前缀“-webkit-”。

13.3 CSS 中为元素应用动画

要在 CSS 中为元素应用动画，首先要创建一个已命名的动画，然后将它附加到该元素属性声明块中的一个元素上。动画本身并不执行任何操作；为了向元素应用动画，需要将动画与元素关联起来。这个要创建的动画，必须使用 @keyframes 来声明（或者对于当前的 Webkit 实现，使用 @-webkit-keyframes），后跟所选择的名称，该名称主要用于对动画的声明作用，然后指定关键帧。

13.3.1 使用 @keyframes 声明动画

一起来看一个来自于 W3C 官网的实例。


```

@keyframes wobble {
  0% {
    margin-left: 100px;
    background: green;
  }
  40% {
    margin-left: 150px;
    background: orange;
  }
  60% {
    margin-left: 75px;
    background: blue;
  }
  100% {
    margin-left: 100px;
    background: red;
  }
}

```

为了节省篇幅，此处没有添加任何浏览器前缀。如果需要对所有支持 @keyframes 的浏览器都有效果，需要添加不同的浏览器前缀，但添加前缀和以往 CSS3 属性添加略有不同。浏览器前缀添加在 keyframes 关键词前面，而不是 @keyframes 前面，如：

- ❑ Firefox 使用 @-moz-keyframes;
- ❑ Chrome 和 Safari 浏览器使用 @-webkit-keyframes;
- ❑ Opera 浏览器使用 @-o-keyframes。

在这个简单的示例中，通过 @keyframes 声明了一个名叫 “wobble” 的动画，它的动画是从 0% 开始到 100% 时结束，同时还经历了一个 40% 和 60% 两个过程。简单而言这个名叫 “wobble” 动画一共具有四个关键帧，实现以下动画效果。

1) “wobble” 动画在 0% (第一帧) 时元素定位到 left 为 100px 的背景色为 green;

2) “40%” (第二帧) 时元素过渡到 left 为 150px 的位置并且背景色为 orange;

3) “60%” (第三帧) 时元素过渡到 left 为 75px 的位置背景色为 blue;

4) “100%” (第四帧) 结束动画的位置元素又回到起点 left 为 100px 处背景变成 red。

假设只给这个动画有 10s 的执行时间，每一段执行的状态如图 13-1 所示。

使用 @keyframes 声明一个动画名，而其

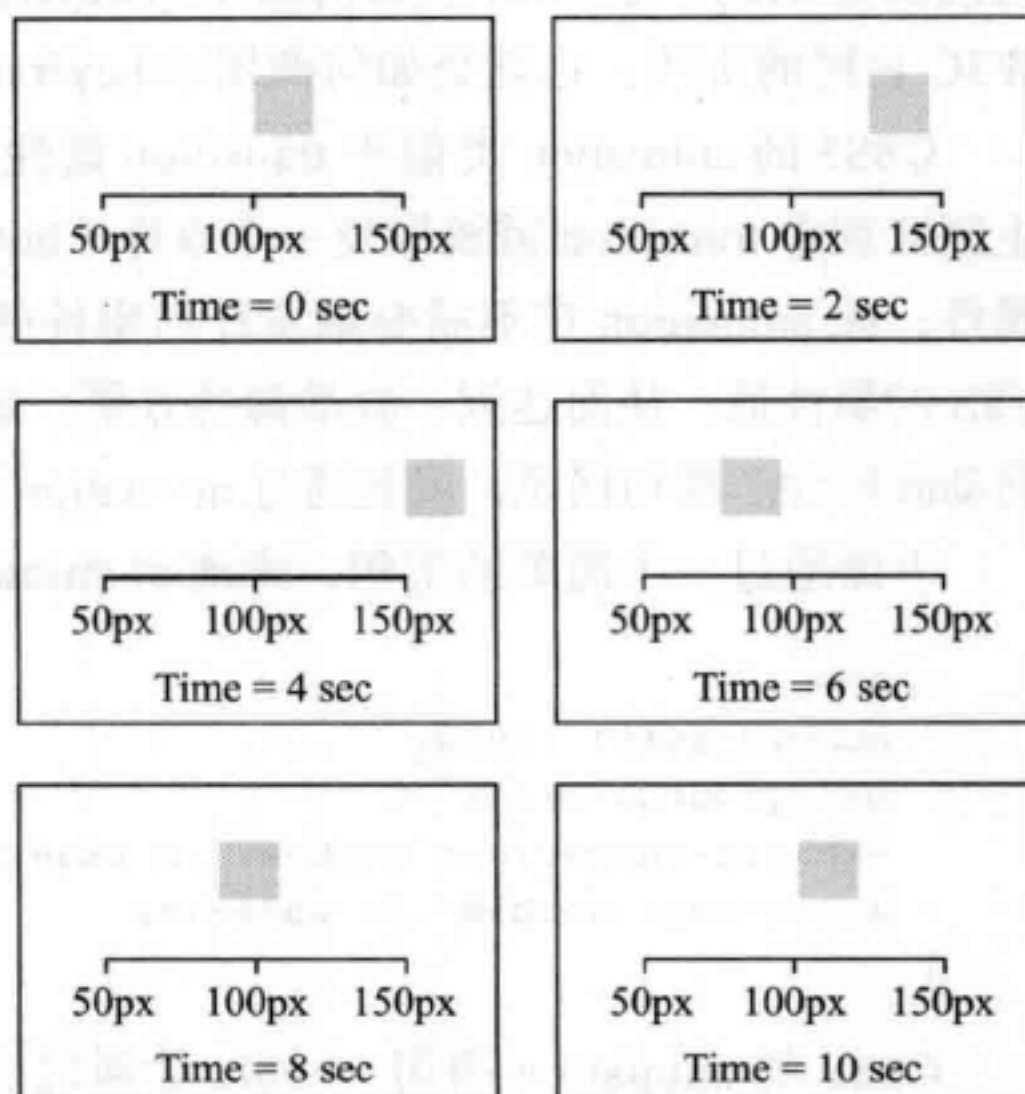


图 13-1 wobble 动画执行示意图

中声明动画的每个关键帧看起来就像它自己嵌套的 CSS 声明块。但是，无需使用选择器，可以使用关键字 `from` 或 `to` 和百分比值，或者一个以逗号分隔的百分比值列表。这些值指定关键帧位于动画中的位置。

在每个关键帧内，包含目标属性和值。在每个关键帧之间，浏览器的动画引擎将平滑地插入值。在 `@keyframes` 中的关键帧并不是一定要按照顺序来指定，其实可以任何顺序来指定关键帧，因为动画中的关键帧顺序由百分比值确定而不是声明的顺序。来看一个简单的例子。

```
@keyframes bounce {
  0%, 20%, 50%, 80%, 100% {transform: translateY(0);}
  40% {transform: translateY(-30px);}
  60% {transform: translateY(-15px);}
}
```

在这个示例中，向 0%、20%、50%、80% 和 100% 五个关键帧中应用了相同的样式。在这种情况下，它意味着元素在 Y 轴方向位移为 0，然后在 40% 和 60% 时元素分别在 Y 轴方向向左移动 30px 和 15px。

在这里演示的两个动画，但它们未附加到任何元素上。这样 `@keyframes` 声明的动画是不起任何作用，不会有任何效果。通过 `@keyframes` 定义动画后，要让 `@keyframe` 声明的动画有效果，将要通过 CSS 属性来调用 `@keyframes` 声明的动画。

13.3.2 调用 @keyframes 声明的动画

`@keyframes` 只是用来声明一个动画，如果不通过别的 CSS 属性调用这个动画，是无任何动画效果的。在 CSS 中如何调用 `@keyframes` 声明的动画呢？一起回到上一节中所示的 W3C 官网的实例，也就是如何调用 `@keyframes` 声明的 `wobble` 动画。

CSS3 的 `animation` 类似于 `transition` 属性，它们都是随着时间改变元素的属性值。它们主要区别是 `transition` 需要触发一个事件（`hover` 事件或 `click` 事件等）才会随时间改变其 CSS 属性；而 `animation` 在不需要触发任何事件的情况下也可以显式地随着时间变化来改变元素 CSS 的属性值，从而达到一种动画的效果。这样一来就可以直接在一个元素中调用 `animation` 的动画属性。换句话说，就是通过 `animation` 属性来调用 `@keyframes` 声明的动画。

下面通过一个简单的实例，来演示 `animation` 属性调用 `@keyframes` 声明的动画。

```
.demo {
  margin-left: 100px;
  background: blue;
  -webkit-animation: wobble .2s ease-in;
  animation: wobble .2s ease-in;
}
```

CSS3 的 `animation` 调用 `wobble` 动画之后，会影响与元素相对应的 CSS 属性值，在整个动画过程中，元素的变化属性值完全是由 `animation` 来控制的，动画后面的值会覆盖前面

的属性值。如前面声明的 wobble 动画。

因为这个 demo 只是在不同的时间段改变了“.demo”元素的背景颜色和距离父元素左边缘的距离，其默认值是“margin-left:100px;background:blue”。

1) wobble 动画执行到 0% 时，元素 demo 的 margin-left 值变成 100px，背景颜色变成 green；

2) 当动画执行到 40% 时，元素 demo 的 margin-left 值变成 150px，背景颜色变成 orange；

3) 当动画执行到 60% 时，元素 demo 的 margin-left 值变成 75px，背景颜色变成 blue；

4) 最后动画执行到 100% 时，元素 demo 的 margin-left 值变成 100px，背景颜色变成 red，同时动画也执行完。

如果未经特殊设置，元素的 margin-left 和 background 值将回到最初默认状态。这也再次证明，动画中设置的属性不会产生叠加效果，只是一次一次覆盖前一次出现的样式。就如 CSS 同一元素样式，最后出现的权限是最大的。

可以看一张来自 W3C 官网的演示 CSS3 的 animation 属性变化过程的示意图，如图 13-2 所示。

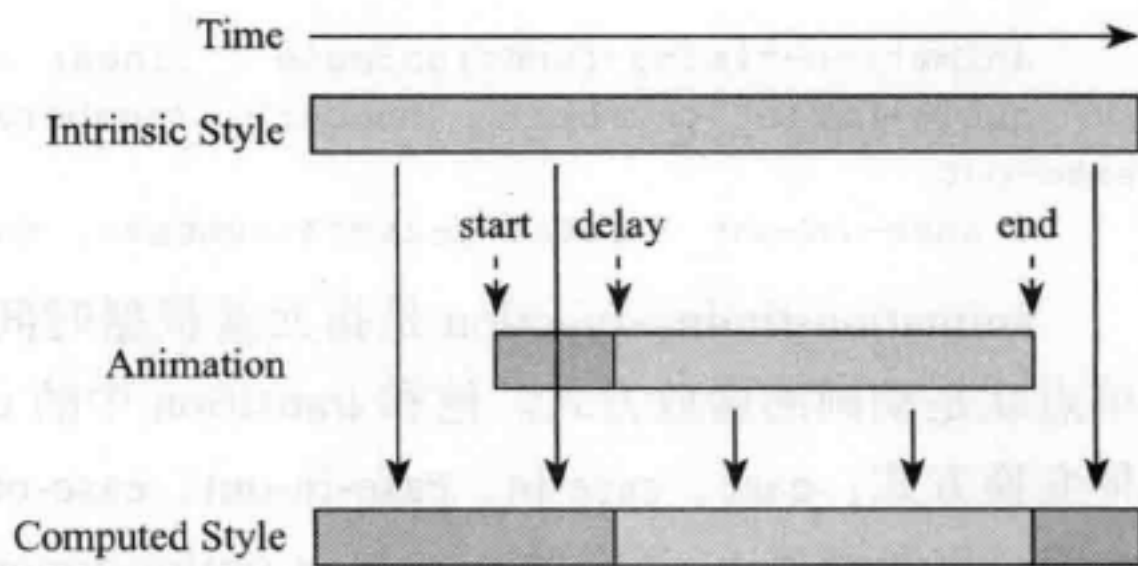


图 13-2 animation 动画演示图

13.4 CSS3 动画子属性详解

CSS3 动画属性 animation 具有八个子属性，其每个子属性所起的作用都不一样，只有了解并掌握了每个子属性的功能以及其使用方法，才能更好地运用 animation 实现一些完美的动画效果。

13.4.1 调用动画 animation-name

animation-name 属性主要是用来调用动画，其调用的动画是通过 @keyframes 关键帧定义好的动画。该属性的使用语法如下。

```
animation-name: none | IDENT[,none|IDENT]*;
```

animation-name 是用来定义一个动画的名称，其主要有两个值。

- IDENT：是由 @keyframes 创建的动画名称，换句话说此处的 IDENT 需要和 @keyframes 中的 IDENT 一致，如果不一致将不能实现任何动画效果。
- none：为默认值，当值为 none 时，将没有任何动画效果，其可以用于覆盖任何动画。

13.4.2 设置动画播放时间 animation-duration

animation-duration 主要用来设置 CSS3 动画播放时间，其基本语法如下。

```
animation-duration: <time>[,<time>]*
```

animation-duration 和 transition-duration 使用方法类似，是用来指定元素播放动画所持续的时间，也就是完成从 0% ~ 100% 一次动画所需时间，取值 <time> 为数值，单位为秒，其默认值为 0，这意味着动画周期为 0，也就是没有动画效果。如果值为负值会被视为 0。

13.4.3 设置动画播放方式 animation-timing-function

animation-timing-function 属性主要是用来设置动画播放方式，其基本语法如下。

```
animation-timing-function: ease | linear | ease-in | ease-out | ease-in-out |  
cubic-bezier(<number>, <number>, <number>, <number>) [, ease | linear | ease-in |  
ease-out  
| ease-in-out | cubic-bezier(<number>, <number>, <number>, <number>)]*
```

animation-timing-function 是指元素根据时间的推进来改变属性值的变换速率，说得简单点就是动画的播放方式。他和 transition 中的 transition-timing-function 一样，具有以下几种变换方式：ease、ease-in、ease-in-out、ease-out、linear 和 cubic-bezier。具体的使用方法大家可以参阅第 12 章中的 transition-timing-function 的使用方法。

13.4.4 设置动画开始播放的时间 animation-delay

animation-delay 属性用来定义动画开始播放的时间、是延迟还是提前等。该属性的基本语法如下。

```
animation-delay: <time>[,<time>]*
```

换句话说，animation-delay 属性用于定义在浏览开始执行动画之前等待的时间。

13.4.5 设置动画播放次数 animation-iteration-count

animation-iteration-count 属性主要用来定义动画的播放次数。其基本语法如下。

```
animation-iteration-count: infinite | <number> [, infinite | <number>]*
```

此属性主要用于定义动画播放多少次，其值通常为整数，但也可以使用带有小数的数字。其默认值为 1，这意味着动画将从开始到结束只播放一次。如果取值为 infinite，动画将会无限次地播放。

13.4.6 设置动画播放方向 animation-direction

animation-direction 属性主要用来设置动画播放方向，其基本语法如下。

```
animation-direction:normal | alternate [, normal | alternate]*
```

animation-direction 是用来指定元素动画播放的方向,其主要有两个值,默认值为 **normal**,如果设置为 **normal** 时,动画的每次循环都是向前播放;另一个值是 **alternate**,它的作用是,动画播放为偶数次则向前播放,为奇数次则向反方向播放。例如一个弹跳动画中,可以为落下的球提供关键帧,然后将 **animation-direction** 取值为 **alternate** 来控制播放动画的每秒中反转它。

13.4.7 设置动画的播放状态 animation-play-state

animation-play-state 属性主要用来控制元素动画的播放状态,其基本语法如下。

```
animation-play-state: running | paused [, running | paused]*
```

animation-play-state 主要有两个值: **running** 和 **paused**。其中 **running** 为默认值,主要作用类似于音乐播放器,可以通过 **paused** 将正在播放的动画停下来,也可以通过 **running** 将暂停的动画重新播放,这里的重新播放不一定是从元素动画的开始播放,也可能是从暂停的那个位置开始播放。另外如果暂停了动画的播放,元素的样式将回到最原始设置状态。

13.4.8 设置动画时间外属性 animation-fill-mode

animation-fill-mode 属性定义在动画开始之前和结束之后发生的操作,其基本语法如下。

```
animation-fill-mode: none | forwards | backwards | both
```

animation-fill-mode 属性主要有四个值: **none**、**forwards**、**backwards** 和 **both**。其默认值为 **none**,表示动画将按预期进行和结束,在动画完成其最后一帧时,动画会反转回到初始帧处。当其取值为 **forwards** 时,动画在结束后继续应用最后关键帧的位置。当其取值为 **backwards** 时,会在向元素应用动画样式时迅速应用动画的初始帧。当其取值为 **both** 时,元素动画同时具有 **forwards** 和 **backwards** 效果。

在默认情况之下,动画不会影响它的关键帧之外的属性,但使用 **animation-fill-mode** 属性,可以修改动画的默认行为。简单地理解就是告诉动画在第一个关键帧上等待动画开始,或者在动画结束时停在最后一个关键帧上而不回到动画第一帧上,或者同时具有这两个效果。

13.5 综合案例:全屏 Slideshow 效果

CSS3 的 **animation** 属性出现之后,很多动画交互效果都可以通过这个属性来实现。在这里介绍一个纯 CSS3 实现全屏幻灯片播放效果。当你点击对应的小图时,会更找一张大图,铺满整个浏览器屏幕,如图 13-3 所示。



图 13-3 CSS3 制作全屏 Slideshow 效果

实现这个效果，首先需要一个好的结构。

```
<section class="demo">
  <div class="slider">
    <ul class="clearfix">
      <li><a href="#bg1">Hipster Fashion Haircut </a></li>
      <li><a href="#bg2">Cloud Computing Services & Consulting</a></li>
      <li><a href="#bg3">My haire is sooo fantastic!</a></li>
      <li><a href="#bg4">Eat healthy & excersice!</a></li>
      <li><a href="#bg5">Lips so kissable I could die ...</a></li>
    </ul>
  </div>
  
  
  
  
  
</section>
```

实现原理，当点击列表中的每个小图时，会将对应的大图铺满整个屏幕。在这个案例中，每个大图出现时，使用了不同的动画效果，它们分别是：图片从左向右出现 (slideLeft)；图片从下向上出现 (slideBottom)；图片从小到大放大出现 (zoomIn)；图片从大到小缩小出现 (zoomOut)；以及图片旋转出现 (rotate)。根据这些动画效果，首先需要给每个动画效果定义一个动画。

```
/* 定义 slideLeft 动画，图片从左向右出现 */
```

```
@keyframes 'slideLeft' {
  0% { left: -500px; }
  100% { left: 0; }
```

```
/* 定义 slideBottom 动画，图片从底部向顶部出现 */
```

```
@keyframes 'slideBottom' {
  0% { top: 350px; }
  100% { top: 0; }
```

```
/* 定义 zoomIn 动画，图片由小到大放大出现 */
```

```
@keyframes 'zoomIn' {
  0% { transform: scale(0.1); }
  100% { transform: none; }
```

```
/* 定义 zoomOut 动画，图片由大到小缩小出现 */
```

```
@keyframes 'zoomOut' {
  0% { transform: scale(2); }
  100% { transform: none; }
```

```
/* 定义 rotate 动画，图片旋转出现 */
```

```
@keyframes 'rotate' {
  0% { transform: rotate(-360deg) scale(0.1); }
  100% { transform: none; }
```

```
}
```


定义好动画之后,需要将图片全屏显示,实现全屏背景方法很多,此处使用最原始的方法,将图片固定定位。

```
html,body {
    height: 100%;
}
img.bg {
    /* 设置全屏填充 */
    min-height: 100%;
    min-width: 1024px;
    width: 100%;
    height: auto !important;
    height: 100%;
    /* 图片定位 */
    position: fixed;
    left: 0;
    z-index: 1;
}
@media screen and (max-width: 1024px) {
    img.bg {
        left: 50%;
        margin-left: -512px; /* 50% */
    }
}
```

接下来,对小图列表简单进行一些样式修饰。

```
.slider {
    position: absolute;
    width: 100%;
    text-align: center;
    z-index: 9999;
    bottom: 100px;
}
.slider li {
    display: inline-block;
    width: 170px;
    height: 130px;
    margin-right: 15px;
}
.slider a {
    display: inline-block;
    width: 170px;
    padding-top: 70px;
    padding-bottom: 20px;
    position: relative;
    cursor: pointer;
    border: 2px solid #fff;
    border-radius: 5px;
    vertical-align: top;
```

```

color: #fff;
text-decoration: none;
font-size: 22px;
font-family: 'Yesteryear', cursive;
text-shadow: -1px -1px 1px rgba(0, 0, 0, 0.8),
             -2px -2px 1px rgba(0, 0, 0, 0.3),
             -3px -3px 1px rgba(0, 0, 0, 0.3);
}
.slider li:nth-of-type(1) a {
    background-color: #02646e;
}
.slider li:nth-of-type(2) a {
    background-color: #eb0837;
}
.slider li:nth-of-type(3) a {
    background-color: #67b374;
}
.slider li:nth-of-type(4) a {
    background-color: #e6674a;
}
.slider li:nth-of-type(5) a {
    background-color: #e61061;
}
.slider a::after {
    content: "";
    display: block;
    height: 120px;
    width: 120px;
    border: 5px solid #fff;
    border-radius: 50%;
    position: absolute;
    left: 50%;
    margin-left: -60px;
    z-index: 9999;
    top: -80px;
}
.slider li:nth-of-type(1) a::after {
    background: url(sbg1.jpg) no-repeat center;
}
.slider li:nth-of-type(2) a::after {
    background: url(sbg2.jpg) no-repeat center;
}
.slider li:nth-of-type(3) a::after {
    background: url(sbg3.jpg) no-repeat center;
}
.slider li:nth-of-type(4) a::after {
    background: url(sbg4.jpg) no-repeat center;
}
.slider li:nth-of-type(5) a::after {
    background: url(sbg5.jpg) no-repeat center;
}

```

```

}
.slider a::before {
    content: "";
    display: block;
    height: 120px;
    width: 120px;
    border: 5px solid #fff;
    border-radius: 50%;
    position: absolute;
    left: 50%;
    margin-left: -60px;
    z-index: 99999;
    top: -80px;
    background: rgba(0,0,0,0.3);
}
.slider a:hover::before {
    opacity: 0;
}

```

现在万事俱备，只欠东风了。整个静态效果完成，但没有动画效果，那是因为没有调用动画。在这个案例中，通过伪类“:target”来调用（开启）刚才定义的动画。

```

.slideLeft:target {
    z-index: 100;
    animation-name: slideLeft;
    animation-duration: 1s;
    animation-iteration-count: 1;
}
.slideBottom:target {
    z-index: 100;
    animation-name: slideBottom;
    animation-duration: 1s;
    animation-iteration-count: 1;
}
.zoomIn:target {
    z-index: 100;
    animation-name: zoomIn;
    animation-duration: 1s;
    animation-iteration-count: 1;
}
.zoomOut:target {
    z-index: 100;
    animation-name: zoomOut;
    animation-duration: 1s;
    animation-iteration-count: 1;
}
.rotate:target {
    z-index: 100;
    animation-name: rotate;
    animation-duration: 1s;
}

```



```
    animation-iteration-count: 1;
}
```

这样一来,就开启了所有的动画,但这样也造成了一个新问题的出现,点击了之后,别的动画就需要关闭,为了实现这个效果,通过“:not(:target)”来完善。

```
@keyframes 'notTarget' {
    0% { z-index: 75; }
    100% { z-index: 75; }
}
.bg:not(:target) {
    animation-name: notTarget;
    animation-duration: 1s;
    animation-iteration-count: 1;
}
```

到此整个效果就出来了,大家可以点击源码中的 DEMO 查看效果。当然除了使用“:target”开启动画之外,更多情况之下是使用单选按钮来开启动画。如果你有兴趣,不妨动手一试。



注意 上面示例中,为了节省篇幅,没有添加各浏览器私有前缀,如果你在使用过的过程中,请添加各浏览器的私有前缀。

13.6 本章小结

早期为了让网站动起来,都依赖于 Flash 或者 JavaScript 来实现。CSS3 的 animation 出现之后,很多效果动画效果完全可以使用 animation 来实现。在本章中详细介绍了 animation 的各个子属性的使用方法。当然,animation 还不是很完美,但通过它可在恰当的地方实现动画效果,以增强用户体验。

媒体特性与 Responsive 设计

随着科学技术不断地向前发展，网页的浏览终端越来越多样化，用户可以通过宽屏电视、台式电脑、笔记本电脑、平板电脑和智能手机来访问网站。尽管无法保证一个网站在不同屏幕尺寸和不同设备上看起来一模一样，但至少要让你的 Web 页面能适配用户的终端，让它更好地呈现在用户面前。在本章中，你将会学到如何使用 CSS3 中的 Media Query 模块来让一个页面适应不同的终端（或屏幕尺寸），从而让页面有一个更好的用户体验。

14.1 媒体类型

媒体类型（Media Type）在 CSS2 中是一个常见的属性，也是一个非常有用的属性，可以通过媒体类型对不同的设备指定不同的样式。

14.1.1 Media Type 设备类型

在 CSS2 中常碰到的就是 all（全部）、screen（屏幕）、print（页面打印或打印预览模式），其实媒体类型远不止这三种，W3C 共列出 10 种媒体类型^①，如表 14-1 所示。

表 14-1 Media Type 设备类型

值	设备类型
All	所有设备
Braille	盲人用点字法触觉回馈设备
Embossed	盲文打印机

① 详细见 <http://www.w3.org/TR/CSS2/media.html#media-types>。

(续)

值	设备类型
Handheld	便携设备
Print	打印用纸或打印预览视图
Projection	各种投影设备
Screen	电脑显示器
Speech	语音或音频合成器
Tv	电视机类型设备
Tty	使用固定密度字母栅格的媒介，比如电传打字机和终端

其中 Screen、All 和 Print 为最常见的三种媒体类型。

14.1.2 媒体类型引用方法

在实际中媒体类型有近十种之多，实际常用的也就那么几种。不过媒体类型的引用方法也有多种，常见的媒体类型引用方法主要有：link 标签、xml 方式、@import 和 CSS3 新增的 @media 几种，下面简单了解这四种引用媒体类型的使用方法。

1. link 方法

link 方法引入媒体类型其实就是在 <link> 标签引用样式的时候，通过 link 标签中的 media 属性来指定不同的媒体类型。这种方式引入媒体类型时常跟着引用的样式文件走，如下所示。

```
<link rel="stylesheet" type="text/css" href="style.css" media="screen" />
<link rel="stylesheet" type="text/css" href="print.css" media="print" />
```

2. xml 方式

xml 方式引用媒体类型和 link 引入媒体类型极其相似，也是通过 media 属性来指定。

```
<?xml-stylesheet rel="stylesheet" media="screen" href="style.css" ?>
```

3. @import 方式

@import 是用来引用样式文件方法之一，同样也可以用来引用媒体类型。@import 引入媒体类型主要有两种方式，一种是在样式中通过 @import 调用另一个样式文件；另一种是在 <head></head> 标签中的 <style></style> 中引入，但这种使用方法在 IE 6 和 IE 7 中都不被支持，如样式文件中调用另一个样式文件时，就可以指定对应的媒体类型。

```
@import url(reset.css) screen;
@import url(print.css) print;
```

在 <head> 中的 <style> 标签中引入媒体类型方法。

```
<head>
  <style type="text/css">
    @import url(style.css) all;
```



```
</style>
</head>
```

4. @media 方式

@media 是 CSS3 中新引进的一个特性，称为媒体查询。在页面中也可以通过这个属性来引入媒体类型。@media 引入媒体类型和 @import 有点类似，也具有两种方式。

1) 在样式文件中引用媒体类型：

```
@media screen {
  选择器 { /* 你的样式代码写在这里... */ }
}
```

2) 使用 @media 引入媒体类型的方式是在 <head> 标签中的 <style> 中引用。

```
<head>
  <style type="text/css">
    @media screen{
      选择器 { /* 你的样式代码写在这里... */ }
    }
  </style>
</head>
```



提示 以上四种方法都可以在页面中引用媒体类型，但这几种方法都有其各自的利弊，在实际应用中个人强烈建议使用第一种和第四种引用媒体类型的方法。因为这两种方式是在项目制作中最常用的方法，对于它们的具体区别，在此处就不用更多的篇幅来阐述了，感兴趣的可以查询相关的文档。

14.2 媒体特性

媒体特性 (Media Query) 是 CSS3 对媒体类型 (Media Type) 的增强版，其实可以将 Media Query 看成 “Media Type (判断条件) + CSS (符合条件的样式规则)”。

14.2.1 Media Query 和 CSS 属性集合

一起来看一个简单的实例。

```
<link rel="stylesheet" media="screen and (max-width:600px)" href="small.css" />
```

通常是按上面的方式引用一个样式的，那么在 CSS3 中可以将上面的形式转换成 CSS。

```
@media screen and (max-width:600px) {
  选择器 { /* 你的样式代码写在这里... */ }
}
```

其实就是把 small.css 样式文件中的样式放在了 @media screen and (max-width:600px){...}

的大括号之中。在上面的代码中可以看出 Media Query 和 CSS 的属性集合很相似，主要区别在如下。

- ❑ Media Query 只接受单个的逻辑表达式作为其值，或者没有值。
- ❑ CSS 属性用于声明如何表现页面的信息；而 Media Query 是一个用于判断输出设备是否满足某种条件的表达式。
- ❑ Media Query 中的大部分接受 min/max 前缀，用来表达其逻辑关系，表示应用于大于等于或小于等于某个值的情况。
- ❑ CSS 属性要求必须有属性值，Media Query 可以没有值，因为其表达式返回的只有真或假两种。

14.2.2 常用 Media Query 设备特性

W3C 共列出来 13 种 CSS3 中常用的特性，如表 14-2 所示。

表 14-2 常用的 Media Query 设备特性

属性	值	Min/Max	描述
color	整数	Yes	每种色彩的字节数
color-index	整数	Yes	色彩表中的色彩数
device-aspect-ratio	整数 / 整数	Yes	宽高比例
device-height	Length	Yes	设备屏幕的输出高度
device-width	Length	Yes	设备屏幕的输出宽度
grid	整数	No	是否基于栅格的设备
height	Length	Yes	渲染界面的高度
monochrome	整数	Yes	单色帧缓冲器中每像素字节
resolution	分辨率 (dpi/dpcm)	Yes	分辨率
scan	Progressive interlaced	No	Tv 媒体类型的扫描方式
width	Length	Yes	渲染界面的宽度
orientation	Portrait/landscape	No	横屏或竖屏

14.2.3 浏览器兼容性

到目前为止，CSS3 Media Query 得到了众多浏览器支持，除了 IE 6 ~ 8 浏览器不支持之外，在所有现代浏览器中都可以完美支持。还有一点与众不同的是，Media Query 不像其他 CSS3 属性一样需在不同的浏览器中添加前缀。

14.2.4 Media Query 使用方法

Media Query 能在不同的条件下使用不同的样式，使页面在不同终端设备下达到不同的渲染效果。前面简单介绍了如何将 Media Query 引用到项目中，但 Media Query 有其自己的使用规则。具体来说，Media Query 的使用方法如下。

```
@media 媒体类型 and (媒体特性) { 你的样式 }
```

使用 Media Query 时必须使用 @media 开头，然后指定媒体类型（也可以称为设备类型），随后是指定媒体特性（也可以称之为设备特性）。媒体特性的书写方式和样式的书写方式非常相似，主要分为两个部分，第一个部分指的是媒体特性，第二部分为媒体特性所指定的值，而且这两个部分之间使用冒号分隔。例如：

```
(max-width: 480px)
```

从表 14-1 和表 14-2 中可以得知，主要有 10 种媒体类型和 13 种媒体特性，将其组合就类似于不同的 CSS 集合。但与 CSS 属性不同的是，媒体特性是通过 min/max 来表示大于、等于或小于作为逻辑判断，而不是使用小于 (<) 和大于 (>) 这样的符号来判断。接下来一起来看看 Media Query 在实际项目中的常用方式。

1. 最大宽度 max-width

“max-width”是媒体特性中最常用的一个特性，其意思是指媒体类型小于或等于指定的宽度时，样式生效，如：

```
@media screen and (max-width:480px) {
  .ads {
    display:none;
  }
}
```

上面表示的是：当屏幕小于或等于 480px 时，页面中的广告区块 (.ads) 都将被隐藏。

2. 最小宽度 min-width

min-width 与 max-width 相反，即媒体类型大于或等于指定宽度时，样式生效。

```
@media screen and (min-width:900px) {
  .wrapper{width: 980px;}
}
```

上面表示的是：当屏幕大于或等于 900px 时，容器 “.wrapper” 的宽度为 980px。

3. 多个媒体特性使用

Media Queries 可以使用关键词 “and” 将多个媒体特性结合在一起。也就是说，一个 Media Query 中可以包含 0 到多个表达式，表达式又可以包含 0 到多个关键字，以及一种媒体类型。

当屏幕在 600 ~ 900px 之间时，body 的背景色渲染为 “#f5f5f5”，如下所示。

```
@media screen and (min-width:600px) and (max-width:900px) {
  body {background-color:#f5f5f5;}
}
```

4. 设备屏幕的输出宽度 Device Width

在智能设备上，例如 iPhone、iPad 等，还可以根据屏幕尺寸来设置相应的样式（或者

调用相应的样式文件)。对于屏幕同样可以使用 min/max 对应参数, 如 min-device-width 或者 max-device-width。

```
<link rel="stylesheet" media="screen and (max-device-width:480px)" href="iphone.css" />
```

上面的代码指的是 iphone.css 样式适用于最大屏幕宽度为 480px, 比如说 iPhone 上的显示, 这里的 max-device-width 所指的是设备的实际分辨率, 也就是指可视面积分辨率。

5. not 关键词

关键词 not 用来排除某种制定的媒体类型, 也就是排除符合表达式的设备。换句话说, not 关键词表示对后面的表达式执行取反操作。如:

```
@media not print and (max-width: 1200px){ 样式代码 }
```

上面代码表示的是, 样式代码将被使用在除打印设备和屏幕宽度小于 1200px 的所有设备中。

6. only 关键词

only 用来指定某种特定的媒体类型, 可以排除不支持媒体查询的浏览器。其实 only 很多时候是用来对不支持 Media Query 却支持 Media Type 的设备隐藏样式表。例如, 支持媒体特性的设备正常调用样式, 此时就当 only 不存在; 不支持媒体特性但支持媒体类型的设备, 这样就不会读样式, 因为其会先读取 only 而不是 screen; 不支持 Media Query 的浏览器, 不论是否支持 only, 样式都不会被采用。如:

```
<link rel="stylesheet" media="only screen and (max-device-width:240px)"
href="android240.css" />
```

在 Media Query 中如果没有明确指定 Media Type, 其默认为 all, 如:

```
<link rel="stylesheet" media="(min-width:701px) and (max-width:900px)"
href="mediu.css" />
```

另外在样式中, 还可以使用多条语句将同一个样式应用于不同的媒体类型和媒体特性中, 指定方式如下:

```
<link rel="stylesheet" type="text/css" href="style.css" media="handheld and
(max-width:480px), screen and (min-width:960px)" />
```

上面代码中 style.css 样式被用在宽度小于或等于 480px 的手持设备上, 或者被用于屏蔽宽度大于或等于 960px 的设备上。

14.3 Responsive 布局概念

随着用户访问 Web 页面终端多样化, 比如说 iPad、iPhone、平板电脑、台式机以及笔记本等, 以前的设计目前再无法适应这些多样化的终端设备。为了满足用户的需求, Ethan

Marcotte 在 A List Apart 发表了一篇开创性的文章，将三种已有的开发技术（弹性网格布局、弹性图片、媒体和媒体查询）整合起来，并将其命名为 RWD（Responsive Web Design，响应式设计）。

什么是响应式设计呢？维基百科是这样描述的：Responsive 设计为 RWD，是精心提供各种设备都能浏览网页的一种设计方法，RWD 能让网页在不同的设备中展现不同的设计风格。由此可见，RWD 不是流体布局，也不是网格布局，而是一种独特的网页设计方法。

14.3.1 Responsive 设计特点

Responsive 网页设计不但要考虑其元素布局的秩序，还要做到“有求必应”，因此需要满足三个条件。Responsive 设计之父 Ethan Marcotte 是这样描述这三个条件的。

- ❑ 网站必须建立灵活的网格基础；
- ❑ 引用到网站的图片必须是可伸缩的；
- ❑ 不同的显示风格，需要在 Media Query 上设置不同的样式。



提示 缺少任何一个功能，就不能称为是合格的 Responsive 网页设计。

14.3.2 Responsive 中的术语

在响应式设计中，有一些其专有的术语，而且理解这些术语对于帮助理解和学习响应式设计至关重要。

1. 流体网格

流体网格是一个简单的网格系统，这种网格设计参考了流体设计中的网格系统，将每个网格格子使用百分比单位来控制网格大小。这种网格系统最大的好处是让网格大小随时根据屏幕尺寸大小做出相对应的比例缩放。

2. 弹性图片

弹性图片指的是不给图片设置固定尺寸，而是根据流体网格进行缩放，用于适应各种网格的尺寸。而实现方法是比较简单，一条代码就能确定。

```
img {max-width:100%;}
```

不幸的是，这条代码在 IE8 浏览器存在一个严重的问题——图片会失踪。当然弹性图片在响应式设计中如何更好地实现，到目前为止，还存在争议，也还在不断地改善。

为每一个断点提供不同的图片，这是一个比较头痛的事情，因为 Media Queries 并不能改变图片“src”的属性值，那有没有办法解决呢？可以参考下面的解决方法。

- ❑ 使用 background-image 给元素添加背景图片。
- ❑ 显示 / 隐藏父元素，给父元素使用不同的图片，然后通过 Media Query 来控制这些图

片显示或隐藏。

来看一个断点解决图片自适应的例子。

```

```

对应的 CSS 代码：

```
@media (min-device-width:600px){
    img[data-src-600px]{
        content: attr(data-src-600px,url);
    }
}
@media (min-device-width:800px){
    img[data-src-800px]{
        content:attr(data-src-800px,url);
    }
}
```

3. 媒体查询

媒体查询在 CSS3 中得到了强大的扩展。使用这个属性可以让设计根据用户终端设备适配对应的样式。这也是响应式设计中最为关键的。可以说，Responsive 设计离开了 Media Query 就失去了它生存的意义。简单地说，媒体查询可以根据设备的尺寸，查询出适配的样式。Responsive 设计最关注的就是：根据用户的使用设备的当前宽度，Web 页面将加载一个备用的样式，实现特定的页面风格。

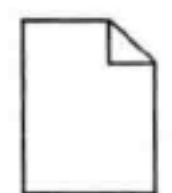
4. 屏幕分辨率

屏幕分辨率指的是用户使用的设备浏览 Web 页面时的分辨率，比如智能手机浏览器、移动电脑浏览器、平板电脑浏览器和桌面浏览器。Responsive 设计利用 Media Query 属性针对浏览器使用的分辨率来适配对应的 CSS 样式。因此屏幕分辨率在 Responsive 设计中是一个很重要的东西，因为只有知道 Web 页面要在哪种分辨率下显示何种效果，才能调用对应的样式。

5. 主要断点

主要断点，在 Web 开发中是一个新词，但其是 Responsive 设计中很重要的一部分。简单的描述就是，设备宽度的临界点。在 Media Query 中，媒体特性 min-width 和 max-width 对应的属性值就是响应式设计中的断点值。简单点说，就是使用主要断点和次要断点，创建媒体查询的条件。而每个断点会对应调用一个样式文件（或者样式代码），如图 14-1 所示。

图 14-1 所示的 style.css 样式文件运用在 Web 页面中，这个样式文件包括了所有风格的样式代码，也就是说，所有设备下显示的风格都通过这个样式文件下载下来。当然，在实际中还可以使用另一种方法，也就是在不同的断点加载不同的样式文件，如图 14-2 所示。



styles.css

```
@media {
  (min-width: 320px)
```

```
@media {
  (min-width: 480px)
```

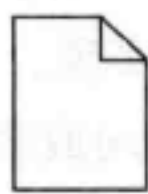
```
@media {
  (min-width: 640px)
```

```
@media {
  (min-width: 768px)
```

图 14-1 主要断点



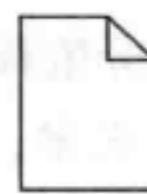
basic.css

(typically)
no media
queries

mobile.css

```
@media {
  (min-width: 480px)
```

```
@media {
  (min-width: 640px)
```



desktop.css

```
@media {
  (min-width: 768px)
```

图 14-2 不同断点加载不同样式文件

图 14-2 所示的是主要断点，主要断点加载样式文件将直接影响 Web 的效果。除了主要断点之外，为了满足更多效果，还可以在这个基础上添加次要断点。不过主要断点和次要断点增加之后，需要维护的样式也相应地增加，成本也相对应地增加。因此在实际使用中，需要根据自身产品需求，决定断点。

综合下来，设置断点应把握以下三个要点。

- ☐ 满足主要的断点；
- ☐ 有可能的话添加一些别的断点；
- ☐ 设置高于 1024px 的桌面断点。

14.3.3 Responsive 布局技巧

布局是一个再简单不过的问题了，也是每个网页设计中必须包含的部分，但使用 Responsive 设计第一步要做的事情是让页面布局尽量简单。实现一个简单的布局有一些小技巧。

- ☐ 尽量少用无关紧要的 div；
- ☐ 不要使用内联元素 (inline)；
- ☐ 尽量少用 JS 或 Flash；
- ☐ 丢弃没用的绝对定位和浮动样式；
- ☐ 摒弃任何冗余结构和不使用 100% 设置。

有舍才有得，哪些东西能帮助 Responsive 确定更好的布局呢？

- ☐ 使用 HTML5 Doctype 和相关指南；

- ❑ 重置好样式 (reset.css);
- ❑ 一个简单的有语义的核心布局;
- ❑ 给重要的网页元素使用简单的技巧, 比如导航菜单之类元素。

使用这些技巧, 无非是为了保持 HTML 简单干净, 而且在页面布局中的关键部分 (元素) 不要过分依赖现代技巧来实现, 比如说 CSS3 特效或者 JS 脚本。

说了这么多, 怎么样的布局或者说 HTML 结构才是简单干净的呢? 这里介绍一个快速测试的方法。

首先禁掉页面中所有的样式 (以及与样式相关的信息), 在浏览器中打开, 如果内容排列有序, 方便阅读, 那么这个结构就不会差到哪里去。

14.3.4 meta 标签

当 Responsive 设计页面在智能设备中进行测试的时候 (比如说 iPhone 或 Android), 会惊奇地发现, 所有的媒体查询都不会生效——页面仍展示为普通的样式, 即一个全局缩小后的页面。

这是因为许多智能手机都使用了一个比实际屏幕尺寸大很多的虚拟可视区域, 主要目的就是让页面在智能手机端阅读时不会因为实际可视区域而变形。为了让智能手机能根据媒体查询匹配对应样式, 让页面在智能手机中正常显示, 特意添加了一个特殊的 meta 标签。这个标签的主要作用就是让智能手机浏览网页时能进行优化, 并且可以自定义界面可视区域的尺寸和缩放级别。

meta 标签使用方法如下。

```
<meta name="viewport" content="" />
```

在 content 属性中主要包括表 14-3 所示属性值, 这些属性值用来处理可视区域。

表 14-3 可视区域 meta 标签的 content 属性值

content 属性值	功能描述
width	可视区域的宽度, 其值可以是一个具体数字或关键词 device-width
height	可视区域的高度, 其值可以是一个具体数字或关键词 device-height
initial-scale	页面首次被显示时可视区域的缩放级别, 取值为 1.0 时将使页面按实际尺寸显示, 无任何缩放
minimun-scale	可视区域的最小缩放级别, 表示用户可以将页面缩小的程度, 取值为 1.0 时将禁止用户缩小至实际尺寸以下
maximun-scale	可视区域的最大缩放级别, 表示用户可以将页面放大的程序, 取值为 1.0 时将禁止用户放大至实际尺寸以上
user-scalable	指定用户是否可以对页面进行缩放, 设置为 yes 将允许缩放, no 为禁止缩放

在实际项目中, 为了让 Responsive 设计在智能设备中能正常显示, 也就是浏览 Web 页面时能适应屏幕的大小并显示在屏幕上, 这可以通过这个可视区域的 meta 标签进行重置, 告诉它使用设备的宽度为视图的宽度, 也就是说, 禁止其默认的自适应页面的效果, 具体

设置如下。

```
<meta name="viewport" content="width=device-width,initial-scale=1.0" />
```

另外，由于 Responsive 设计只有结合 CSS3 的 Media Queries 属性，才能尽显 Responsive 设计风格，故浏览器必须支持 CSS3 Media Queries。但大家都清楚，在 IE 6 ~ 8 中是不支持 CSS3 Media Queries 的。那么为了让 IE 6 ~ 8 支持，就很有必要在 IE 9 以下的浏览器中加上以下脚本。

❑ media-queries.js(<http://code.google.com/p/css3-mediaqueries-js/>)

❑ respond.js(<https://github.com/scottjehl/Respond>)

```
<!--[if lt IE9]>
  <script src=http://css3-mediaqueries-js.googlecode.com/
    svn/trunk/css3-mediaqueries.js></script>
<![endif]>
```

14.4 本章小结

本章学习了什么是 CSS3 媒体查询，如何在 CSS 文件中引入媒体查询，如何使用媒体查询来制作响应式设计，以及如何让所有页面适配不同的用户终端设备。

嵌入 Web 字体

一直以来 Web 设计师在设计网页时都想为 Web 页面添加一些优雅的字體，但浏览器仅限于使用用户在其系统上安装的字體呈现文本，这样一来让大部分网站受限于字體数量的不足。多年来一直都是使用图片替换文本的方式来解决页面上使用优雅字體，但这种对于频繁更换文本的网站来说是一件不切实际的事，以致于我们坚持使用这些少量的 Web 字體。随着技术的不断发展，出现在 Web 页面中使用 Flash 和 JavaScript 技术来弥补这一不足。虽然这些方法已经是不错的应急措施，允许包含自己的字體，但是它们拥有很严重的缺陷。有时候它们很难实现，因为它们要求用户在本地启用 JavaScript，或者 Flash 插件。

值得庆幸的一件事情是，可以使用 `@font-face` 模块来解决 Web 页面中使用优雅字體的方式。本章将一起探讨如何在 Web 页面中使用 `@font-face` 来嵌入字體，以实现在 Web 页面中使用优雅字體的目的。

15.1 @font-face 模块介绍






`@font-face` 主要是把自己定义的字體嵌入到 Web 页面中，这些自定义的字體被放置在服务器上，浏览器会根据指定的命令将对应的字體下载到本地缓存，使用它来修饰文本。也常常把这种方式称为 Web 字體嵌入（实际上并没有什么字體被嵌入）。

15.1.1 浏览器兼容性

早在 1998 年 `@font-face` 就被写入了 CSS2 中，但是在 CSS2.1 中又被移出，幸好在 CSS3 中，`@font-face` 成为 CSS3 中的一个模块。`@font-face` 在众多浏览器得到完美支持，

包括 IE 低版本浏览器。另外，@font-face 属性无须添加任何浏览器的私有前缀。

表 15-1 @font-face 浏览器兼容性

属性名称					
@font-face	5.5+ ✓	3.5+ ✓	4.0+ ✓	3.2+ ✓	10.0+ ✓

15.1.2 @font-face 语法

有了 @font-face 模块，只要将字体传入到服务器端，不管用户端是否安装了对应的字体，设计的网页都能够正确显示。用较为专业的话来讲，@font-face 能够加载服务器端的字体，让客户端浏览器显示客户端没有安装的字体。如果没有 @font-face，客户端浏览器只能在客户系统中寻找指定的字体，这样一来给设计师带来极大的限制。

@font-face 能加载服务器端的字体，让客户端浏览器寻找到对应的字体，使用中具有一套成熟的语法规则。

```
@font-face {
  font-family:<YourWebFontName>;
  src: <source>[<format>][,<source>[<format>]]*;
  [font-weight:<weight>];
  [font-style:<style>];
}
```

取值说明如下。

- ❑ YourWebFontName：指定的是自定义的字体名称，最好是使用下载的默认字体文件名，它将被引用到 Web 元素中的 font-family。如 “font-family:'YourWebFont-Name'”。
- ❑ Source：指定的是自定义的字体存放路径，可以是相对路径也可以是绝对路径。
- ❑ Format：指定的是自定义的字体格式，主要用来帮助浏览器识别，其值主要有以下几种类型，如 truetype、opentype、truetype-aat、embedded-opentype、avg 等。
- ❑ font-weight 和 font-style：前者用来指定字体是否为粗体，后者主要定义字体样式，如斜体。除了这两个属性之外，类似的属性还有 font-variant、font-size、font-stretch 等。

15.1.3 使用字体图标的优势

使用字体图标和位图，它自身更具有哪些优势呢？

- ❑ 适用性：一个图标字体比一系列的图像（特别是在 Retina 屏中使用双倍大小的图像）要小。一旦图标字体加载了，图标就会马上渲染出来，不需要下载任何图像。
- ❑ 可扩展性：图标字体可以用过 font-size 属性设置大小。能够随时输出不同大小的图标，然而，使用位图必须为每个不同大小的图像输出一个不同文件。

- 灵活性：文字效果可以很容易地应用到图标上，包括颜色、阴影和翻转等效果。它们还可以在任何背景下显示。
- 兼容性：网页字体支持所有现代浏览器，包括低版本 IE。

15.2 实现 @font-face

@font-face 属性和 CSS3 中的 @media、@import、@keyframes 等属性一样，都是用关键字符 “@” 封装多项规则。@font-face 的 @ 规则主要用于指定自定义字体，然后在其他样式块中调用 @font-face 中自定义的字体。

15.2.1 使用 @font-face 自定义字体

正常使用 @font-face 自定义字体，必须满足以下几个关键点。

- 将各种格式字体上传到服务器上，以支持各种浏览器；
 - 在 @font-face 中显式指定自定义字体名称以及引用自定义字体的字体来源。
- 来看一个 @font-face 规则自定义字体的示例。

```
@font-face {
  font-family: "NeuesBauenDemo";
  src: url("../fonts/neues_bauen_demo-webfont.eot");
  src: url("../fonts/neues_bauen_demo-webfont.eot?#iefix") format("embedded-opentype"),
    url("../fonts/neues_bauen_demo-webfont.woff") format("woff"),
    url("../fonts/neues_bauen_demo-webfont.ttf") format("truetype"),
    url("../fonts/neues_bauen_demo-webfont.svg#NeuesBauenDemo") format("svg");
}
```

代码块中的 font-family 和 src 都是必需的，通过 font-family 来自定义字体，而 src 是引用自定义字体的来源。当然，除了这两个属性之外，还可以在 @font-face 显式地通过字体相关属性设置文本样式，如 font-weight、font-style 等。

@font-face 规则中的 font-family 与其他样式中的 font-family 略有不同。在 @font-face 中的 font-family 只是声明了字体的名称（也就是自定义了字体的名称），而没有向元素中分配这种字体。而样式中的 font-family 却是显式地为元素指定字体名称。

在 @font-face 规则中通过 font-family 来自定义字体名称，而这个字体名称可以是任意的名称或形式，它仅用于元素样式中的 font-family 属性引用。当然，使用的字体名称最好与引用的字体文件名相同，用于保持 CSS 的可读性，可维护性。

上面通过 @font-face 声明了字体名 “NeuesBauenDemo”，但并不会有任何实际效果，如果想让 Web 中的文本字体是 NeuesBauenDemo，需要在样式代码块中的对应元素中引用 @font-face 定义好的字体，如：


```
h2 {font-family:"NeuesBauenDemo";}
```

其效果如图 15-1 所示。

Neues Bauen Demo

图 15-1 NeuesBauenDemo 字体运用

@font-face 和 @keyframes 一样，一个 @font-face 规则仅自定义一个字体，如果需要自定义多个字体就需要对应启用多个 @font-face 规则，如：

```
@font-face {
    font-family:"YourWebFont";
    ...
}
@font-face {
    font-family: "YourWebFont2"
    ...
}
```

15.2.2 声明字体来源

@font-face 规则中有一个非常重要的参数，就是 src，这个属性类似于 img 标签中的 src 属性，其值主要是用于链接到实际的字体文件。此外，可以声明多个来源，如果客户端的浏览器未能找到第一个来源，它会依次尝试寻找后面字体来源，直到找到一个可用的字体来源为止，如：

```
@font-face {
    font-family:"NeuesBauenDemo";
    src:url("../fonts/neues_bauen_demo-webfont.eot");
    src:url("../fonts/neues_bauen_demo-webfont.eot?#iefix") format("embedded-opentype"),
    url("../fonts/neues_bauen_demo-webfont.woff") format("woff"),
    url("../fonts/neues_bauen_demo-webfont.ttf") format(truetype),
    url("../fonts/neues_bauen_demo-webfont.svg#NeuesBauenDemo") format("svg");
}
```

上面的代码块中依次声明了四种字体：EOT、WOFF、TTF 和 SVG。每种字体都有其具体作用。

1. TureType(.ttf) 格式

TureType(.ttf) 格式字体是 Windows 和 iOS 的最常见的字体，是一种 RAW 格式。支持这种字体的浏览器有 IE 9+、Firefox 3.5+、Chrome 4+、Safari 3+、Opera 10+、iOS Mobile Safari 4.2+ 等。

2. OpenType(.otf) 格式

OpenType(.otf) 格式字体被认为是一种原始的字体格式，其内置在 TureType 的基础上，

所以也提供更多的功能，支持这种字体的浏览器有 Firefox 3.5+、Chrome 4.0+、Safari 3.1+、Opera 10.0+、iOS Mobile Safari 4.2+ 等。

3. Web Open Font Format(.woff) 格式

Web Open Font Format(.woff) 格式字体是 Web 字体中最佳格式，它是一个开放的 TrueType/OpenType 的压缩版本，同时也支持元数据包的分离，支持这种字体的浏览器有 IE 9+、Firefox 3.5+、Chrome 6+、Safari 3.6+、Opera 11.1+ 等。

4. Embedded Open Type(.eot) 格式

Embedded Open Type(.eot) 格式字体是 IE 专用字体，可以从 TrueType 中创建此格式字体，支持这种字体的浏览器有 IE4+。

5. SVG(.svg) 格式

SVG(.svg) 格式字体是基于 SVG 字体渲染的一种格式，支持这种字体的浏览器有 Chrome 4+、Safari 3.1+、Opera 10.0+、iOS Mobile Safari 3.2+ 等。

这就意味着在 @font-face 中至少需要 “.woff” 和 “.eot” 两种格式字体，甚至还需要 “.svg” 等字体以得到更多种浏览版本的支持。为了使 @font-face 得到更多的浏览器支持，Paul Irish 写了一个独特的 @font-face 语法叫做 “Bulletproof @font-face”^①，如：

```
@font-face {
  font-family: "YourWebFontName";
  src: url("YourWebFontName.eot?") format("eot"); /*IE*/
  src: url("YourWebFontName.woff") format("woff"),
  url("YourWebFontName.ttf") format("truetype"); /*non-IE*/
}
```

添加这些额外的字体格式可确保支持每种浏览器。但是，在 IE 9 之前版本的浏览器存在一个问题。这些浏览器将第一个 URL 和最后一个之间的所有内容视为一个 URL，以至于无法加载字体。为了解决这个问题，FontSpring 在一篇文章^②中介绍一种方法：在 “.eot” 字体后添加查询字符串。这样使浏览器认为 src 属性的剩余部分是查询字符串的延续，因此可以让浏览器找到正确的 URL，并加载字体。

```
@font-face {
  font-family: "YourWebFontName";
  src: url("YourWebFontName.eot?#iefix") format("eot"); /*IE*/
  src: url("YourWebFontName.woff") format("woff"),
  url("YourWebFontName.ttf") format("truetype"); /*non-IE*/
}
```

这虽然使 IE 9 以下版本浏览器能正常加载所需的字体，但在 IE 9 浏览器的兼容模式之下不会正常加载 EOT 格式字体。换言之，在 IE 9 浏览器的兼容模式下加载 EOT 格式字

① 详见 <http://paulirish.com/2009/bulletproof-font-face-implementation-syntax/>。

② 详见 <http://www.fontspring.com/blog/the-new-bulletproof-font-face-syntax>。

体会出错。要解决此问题，需要在 IE 9 浏览器的兼容模式下添加一个 src 值。

```
@font-face {
  font-family:"YourWebFontName";
  src:url(YourWebFontName.eot);/*IE9 兼容模式 */
  src:url("YourWebFontName.eot?#iefix") format("eot");/*IE*/
  src:url("YourWebFontName.woff") format("woff"),
  url("YourWebFontName.ttf") format("truetype");/*non-IE*/
}
```

除此之外，为了能让更多的浏览器支持，特别让移动设备的浏览器能兼容，需要添加更多的字体。

```
@font-face{
  font-family:"YourWebFontName";
  src:url("YourWebFontName.eot");/*IE9 兼容模式 */
  src:url("YourWebFontName.eot?#iefix") format("embedded-opentype"),/*IE6 ~ IE8*/
  url("YourWebFontName.woff") format("woff"),/* 现代浏览器 */
  url("YourWebFontName.ttf") format("truetype"),/*Safari,Android,iOS*/
  url("YourWebFontName.svg# YourWebFontName") format("svg");/*Legacy iOS*/
}
```

15.2.3 创建各种字体

通过前面的介绍，在 @font-face 中，使用了四种字体格式 EOT、WOFF、TTF 和 SVG。但很多情况之下，手上仅有一种格式的字体，而为了兼容浏览器各版本，需要从一种格式字体转换成所需的各种字体格式。那么如何将字体转换为所有格式的字体呢？这将成为 @font-face 定义字体之前首要解决的问题之一。

目前为止，在互联网上有很多在线转换字体的生成工具。

❑ FontSquirrel 字体转换生成器^①

❑ Codeandmore 字体转换生成器^②

接下来以 Font Squirrel 这个简单的工具为例，向大家介绍如何将一种字体转换为浏览器支持的 @font-face 的所有字体。

1. 获取特殊字体

我们拿下面这种 single Malta 字体为例，如图 15-2 所示。图 15-2 single Malta 特殊字体

要得到“single Malta”字体，不外乎两种途径，其一找到付费网站购买字体；其二就是到免费网站下载字体（为了避免法律纠纷，最好获取得到许可字体）。在 Google Web Fonts^③和 Dafont.com^④等在线网站上都可以下载到许可的免费特殊字体。此处的“single

SINGLE MALTA

① <http://www.fontsquirrel.com/tools/webfont-generator>。

② <http://fontface.codeandmore.com/>。

③ <http://www.google.com/webfonts>。

④ <http://www.dafont.com/>。

Malta”字体来自于 Dafont.com 网站^①，如图 15-3 所示。

点击“下载”按钮下载所需的特殊字体 single malta，并解压缩下载的字包，如图 15-4 所示。

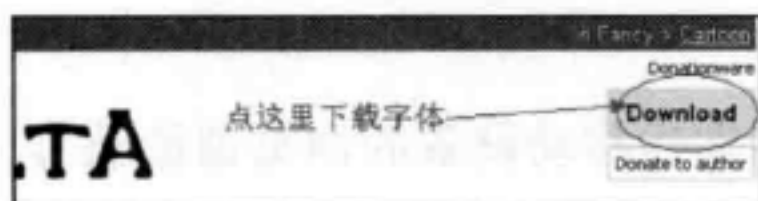


图 15-3 下载 single Malta 特殊字体



图 15-4 解压缩字体包

2. 获取 @font-face 所需字体格式

特殊字体已经在电脑中了，而仅有字体格式是“.ttf”，但 @font-face 所需的“.eot”、“.woff”和“.svg”字体格式，需要使用第三方工具或者软件来实现，接下来使用 FontSquirrel 将“.ttf”字体格式转换成所需的字体。

首先登录 FontSquirrel 在线转换工具^②的页面，将会看到图 15-5 所示界面。

点击“Add Fonts”按钮，添加特殊字体，如图 15-6 所示。

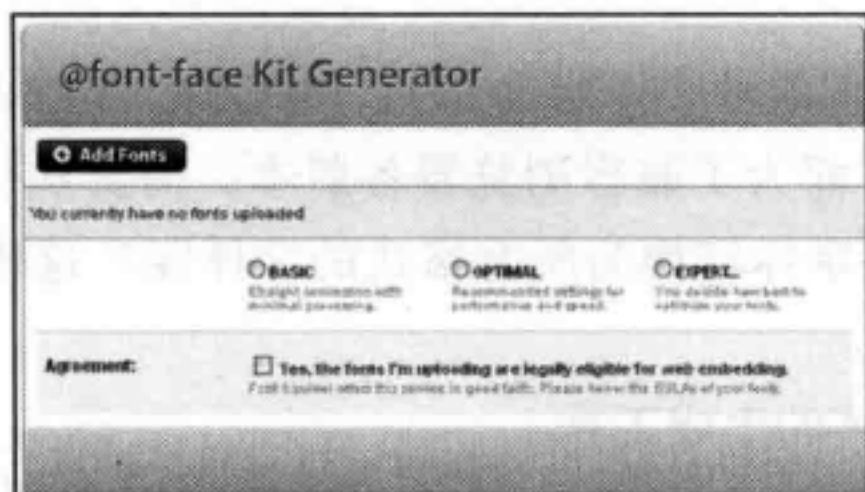


图 15-5 @font-face Kit Generator 工作界面



图 15-6 添加字体文件

上传完字体文件之后，按图 15-7 所示进行操作，将获取所有的字体格式。



图 15-7 转换特殊字体

① <http://www.dafont.com/single-malta.font>。

② 详情见 <http://www.fontsquirrel.com/fontface/generator>。

在本地电脑中可以看到从 FontSquirrel 下载下来的文件，接着只需要对它进行解压缩，就能看到 @font-face 规则所需的所有文件格式。

解压缩出来的文件列表中除了 @font-face 所需要的字体格式之外，还带有一个 DEMO 文件，如果不清楚如何使用 @font-face，可以参考下载下来的 DEMO 文件。

现在 @font-face 所需字体已经加载到本地项目，在本地项目中的 style.css 中附上所需的 @font-face 样式代码块。

```
@font-face {
  font-family: 'SingleMaltaRegular';
  src: url('../fonts/singlemalta-webfont.eot');
  src: url('../fonts/singlemalta-webfont.eot?#iefix') format('embedded-opentype'),
        url('../fonts/singlemalta-webfont.woff') format('woff'),
        url('../fonts/singlemalta-webfont.ttf') format('truetype'),
        url('../fonts/singlemalta-webfont.svg#SingleMaltaRegular') format('svg');
  font-weight: normal;
  font-style: normal;
}
```



图 15-8 解压缩字体

15.2.4 调用字体

到这里为止，已经通过 @font-face 自定义好所需的 single Malta 字体，离最后效果仅一步之差，就是把自己定义的字体应用到 Web 的 DOM 元素上。

```
h2{font-family:"SingleMaltaRegular";}
```

15.3 综合案例：将图标转换成 Web 字体

在这个案例中，将使用一个免费的 Web 应用程序 IcoMoon^① 将矢量图转换成 Web 字体，然后将生成的字体通过 @font-face 应用到 Web 页面中。

15.3.1 创建一个图标字体

Symbol 字体可以使用一个专用的字体创建应用程序，比如说 Glyphs^②，但是一个专业的排版工具之外的需求或要求构建一个简单的图标字体，比如说间中攻粗细这样的物理关系并不是非常重要。

目前为止，最简单的方法是使用 Keyamoon 制作的一个 Web 应用程序 IcoMoon，可以解决字符转换成 Web 字体的所有麻烦。

① 详情见 <http://icomoon.io/app/>。

② 详情见 <http://glyphsapp.com/>。

IcoMoon 附带了大量的图标，可以通过图标库添加更多的图标，其中大部分都可以免费使用（使用时请先查看它们的许可证）。如果你正在寻找如“文件下载”和“购物车”一样的图标，那么你会发现，使用标准的图标比你自己创建要方便得多。

15.3.2 准备插图^①

首先，需要能创建矢量图标的程序，并且能够找到输出 SVG 格式，比如 Illustrator 或者 iNkscape。设计的时候，可以使用任何你喜欢的颜色，但是图标必须是一个纯色。确保每个图标的尺寸大小是相同的。若某一个图标相较其他的更高或者更宽，会导致很难创建一个一致的字体。在这里，我们不得不减少飞艇图标的宽度，以使它能匹配其他图标，如图 15-9 所示。

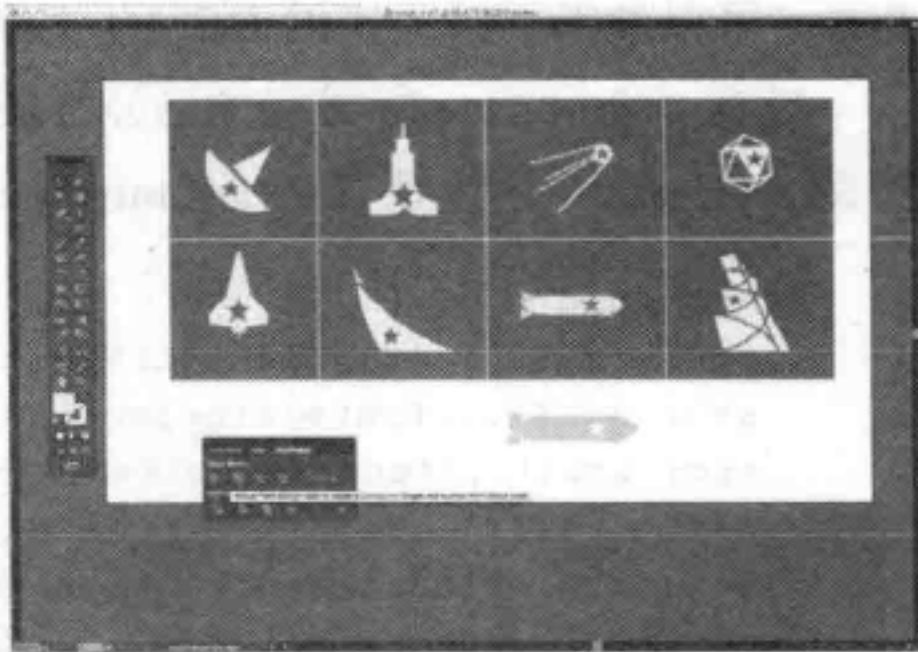


图 15-9 准备插图

1. 清理图标

仔细检查每一个图标，以确保它们没有缺陷——图标在小尺寸的时候可能看着是完美的，当放大后会发现一些小的缺陷。

在 Illustrator 中，使用 Pathfinder 工具统一层叠元素，减去前面元素，比如图标中的星星图标。原则是确保图标是可读的小尺寸，如图 15-10 所示。

2. 导出 SVG 文件

现在，选择一个图标并将它复制粘贴到一个新的文档场景中（如 200px × 200px）。会发现它有一个基线尺寸的设置。使用彩色的图标，比如说在白色的背景中使用黑色的图标。

现在，选择菜单“文件”中“保存”，将选择将文件保存成 SVG 文件格式。使用默认的 SVG 设置。一旦这样做，所有的图标可以创建为一个 Web 字体，如图 15-11 所示。

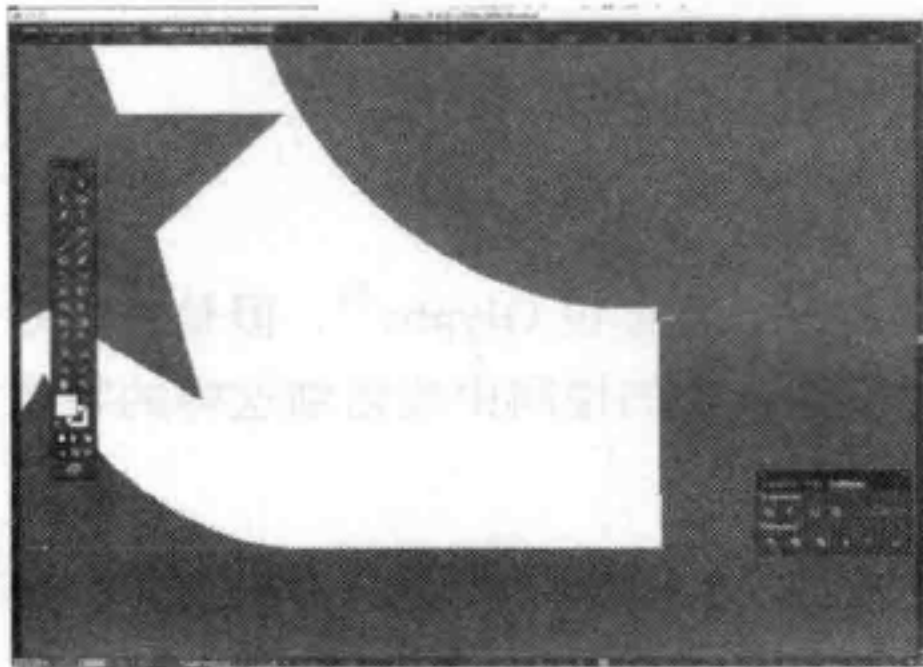


图 15-10 清理图标

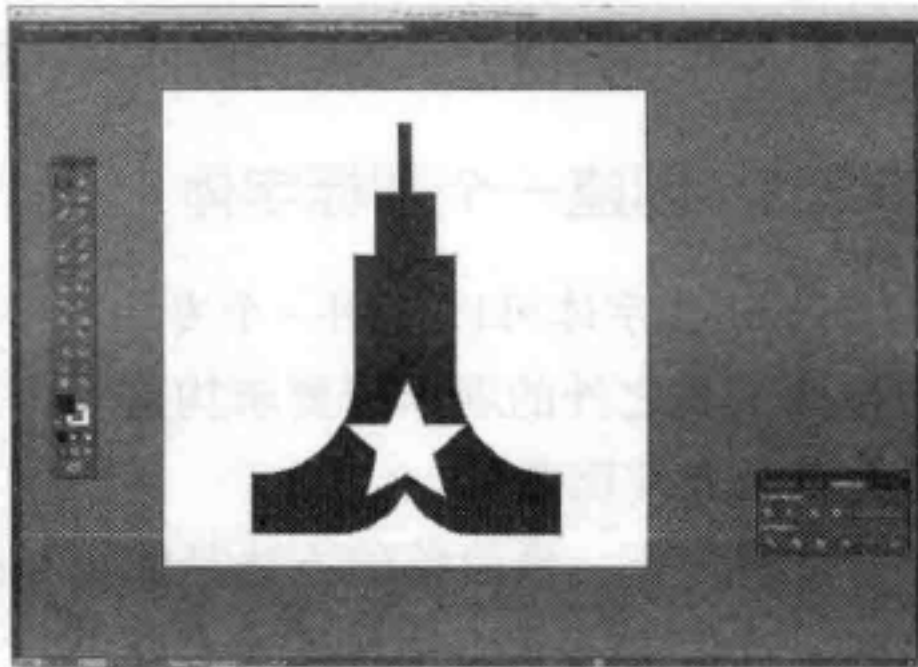


图 15-11 导出 SVG 文件

① 图 15-9 ~ 图 15-14 以及案例实战中的思路来自于 <http://www.webdesignerdepot.com/2013/04/how-to-turn-your-icons-into-a-web-font/>。

15.3.3 导入到 IcoMoon

打开 IcoMoon Web App^①。导入一个图标，点击“Import icons”按钮，然后选择想要添加的 SVG 文件——可以一次添加多个文件。这些图标将会出现在“Your Custom Icons”区域中。如果它们是高亮的黄色显示，表示这些图标是将要创建的图标字体。在这个例子中可以看到，不仅导出了创建的图标，还在“Mini-icons”中添加了别的图标，如图 15-12 所示。

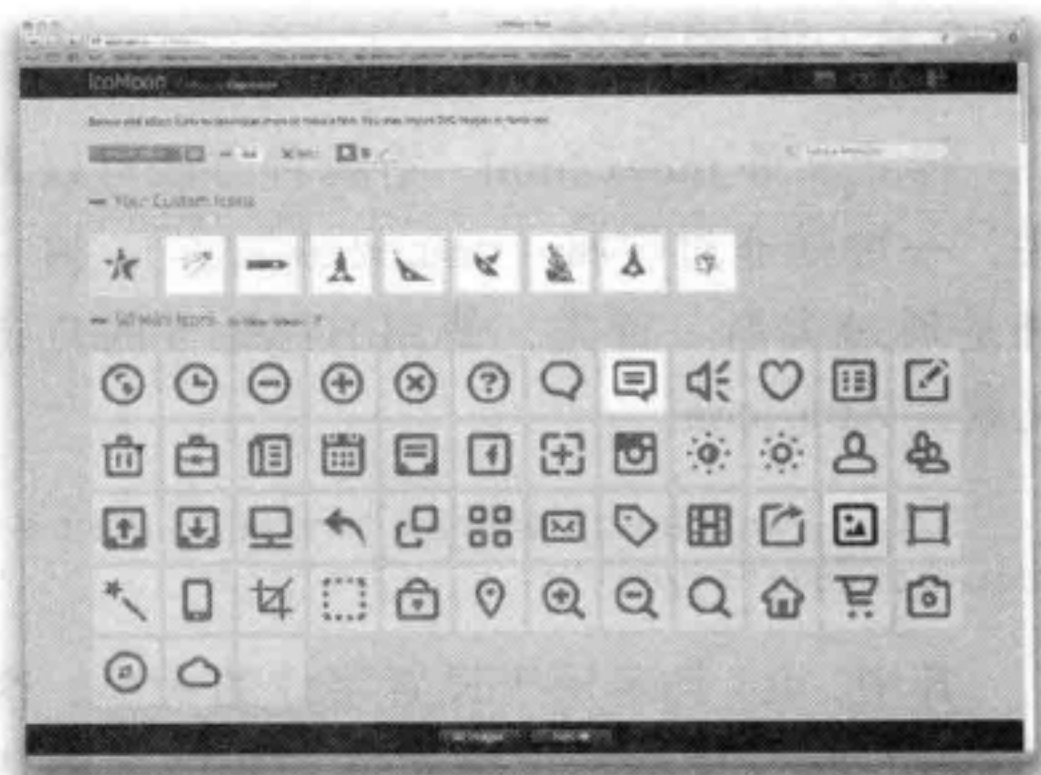


图 15-12 导入 IcoMoon

15.3.4 从 IcoMoon 中导出字体

想调整图标的位置、大小或旋转，可以点击“Edit”按钮，使用“Save Copy”保存图片（如镜像图标）。然后添加一个有意义的图标标记，将用来生成类名。

都准备好了，点击屏幕底部的“Font”按钮开始生成字体。这样就可以指定图标映射到字符上，例如，设置一套六个旋转的球，可以给这六个球分别指定字符 q、w、e、r、t 和 y。也可以根据自己的爱好选择一个字体的名字。导出图标如图 15-13 所示。

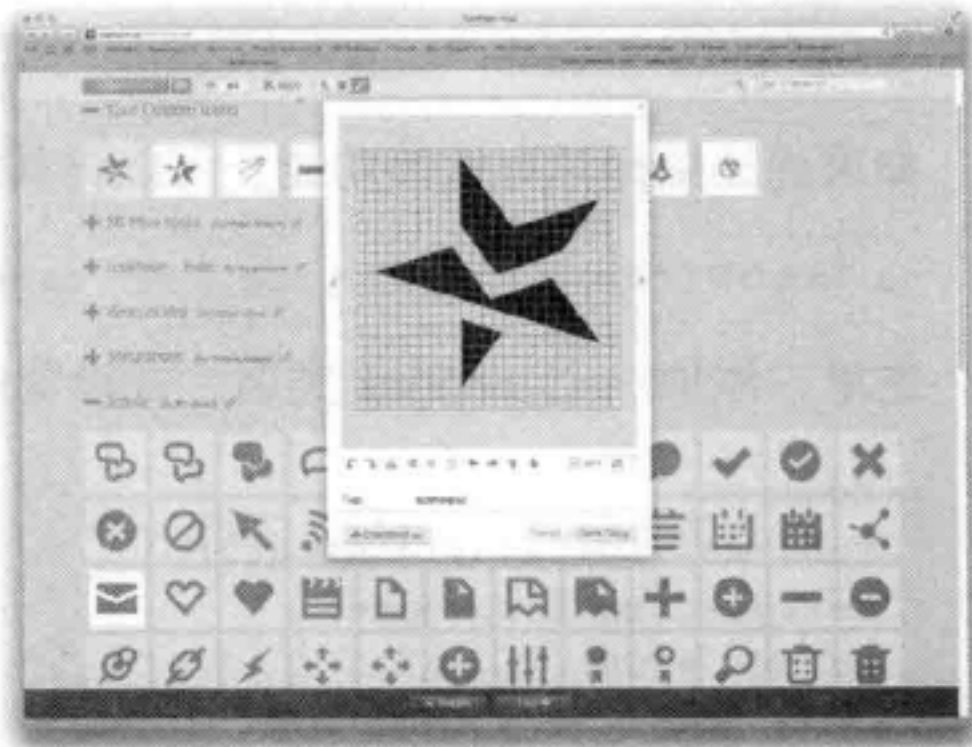


图 15-13 从 IcoMoon 中导出字体

15.3.5 下载字体文件

单击“Download”下载字体包到电脑上。它有一个文件夹包含了字体本身（woff、eot、ttf 格式），以及一个 HTML 示例页面和相应的 CSS。甚至还有一个 JavaScript 文件和一个解决方法，如果需要使用 IE 或 IE7，将 font 文件夹复制到网站，为项目添加字体。需要从 style.css 文件中复制 CSS 样式，并粘贴到自己网站的 CSS 文件中，但是可以要将它重命名为 fonts.css，并保持它作为一个单独的 CSS 文件。需要的时候再把这个 CSS 文件引入到 HTML 中。

```
<link rel="stylesheet" href="fonts.css" />
```

在 CSS 文件中可以找到 @font-face 需要将 URL 路径修改成本地的相对路径，或者可

① 详情见 <http://icomoon.io/app>。

以简单地把字体文件和样式放在同一个文件夹，如图 15-14 所示。

15.3.6 调用字体

样式文件 index.html，有两种方法可以调用，一种是通过字符（unicod 或名称），另一种是通过类名。首先，使用 HTML 5 的自定义属性 data-icon。

```
<div aria-hidden="true" data-icon="&#x67;"></div>
```

其中，fs1 类名用于设置字体的大小。这个 aria-hidden 属性确保字符能利用屏幕阅读器读到。

第二种方法使用一个 span 元素。

```
<span aria-hidden="true"></span>
```

如果想让图标具有链接功能，可以将其放在一个链接中。

```
<a href="http://www.w3cplus.com" data-icon="&#x73;"></a>
```

这里，添加一个 iconlink 类名，并设置一个悬浮效果。

```
.iconlink{
  font-family:"youriconfont";
  text-decoration: none;
  color: #666;
}
.iconlink:hover {
  text-decoration: none;
  color: #999;
}
```

正如悬停状态下改变图标颜色一样，可以使用颜色属性和字体属性修改图标。可以设置其他属性，比如文本阴影和透明度来设置图标的效果。

15.4 本章小结

众所周知，Web 页面的字体运用受到很多限制，浏览器仅限于使用用户在其系统上安装的字体呈现文本。今天，CSS3 使用 @font-face 将改变这一格局。本章详细介绍了使用 @font-face 规则声明自定义字体块，实现特殊字体在 Web 页面中的使用。

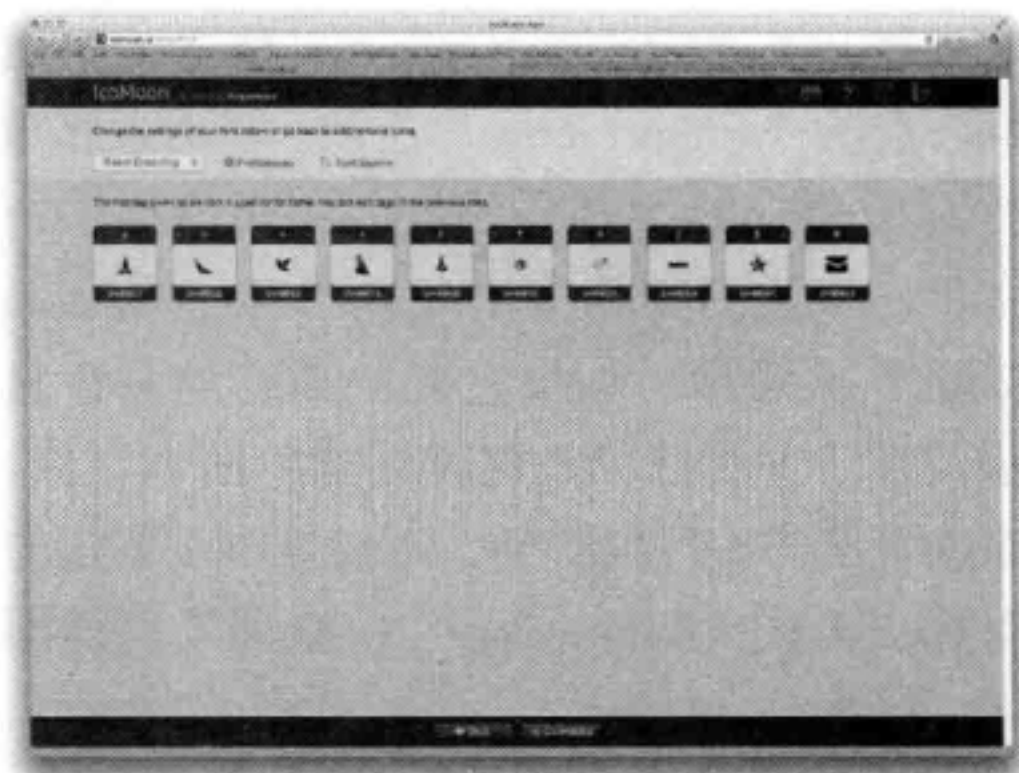
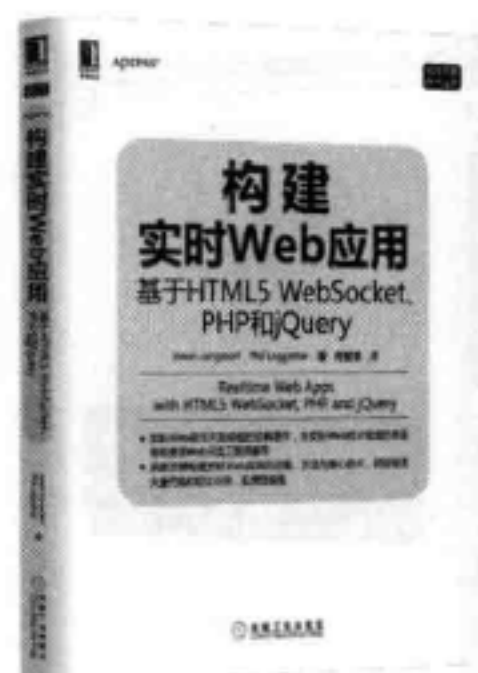


图 15-14 下载字体文件

Web前端开发&设计经典

HTML5&CSS3篇



Web前端开发&设计经典

框架篇

